RasPi

DESIGN
BUILD
CODE

35

Get hands-on with your Raspberry Pi

CONTROL A ROBOT ARM

FIND YOUR **PHONE** WITH PI

Plus Program a family calendar with Python

Welcome

Between doing all the coding and hacking the hardware, working on a Raspberry Pi project can sometimes leave you feeling overwhelmed. Well, this month's main feature is here to offer a helping hand – in the form of a robotic arm. Read on to find out how to build and program this gadget, which can pick up and move items up to 100g in weight.

If you're still in need of extra assistance, this month we'll also show you to how to use your microcomputer to find your missing phone and program a calendar to keep track of all of your family member's birthdays. Our experts also solve many of your project problems in our Talking Pi section. If you have a similar Pi query, don't forget you can always reach us via the below contact details.

Jack Parsons
Editor

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

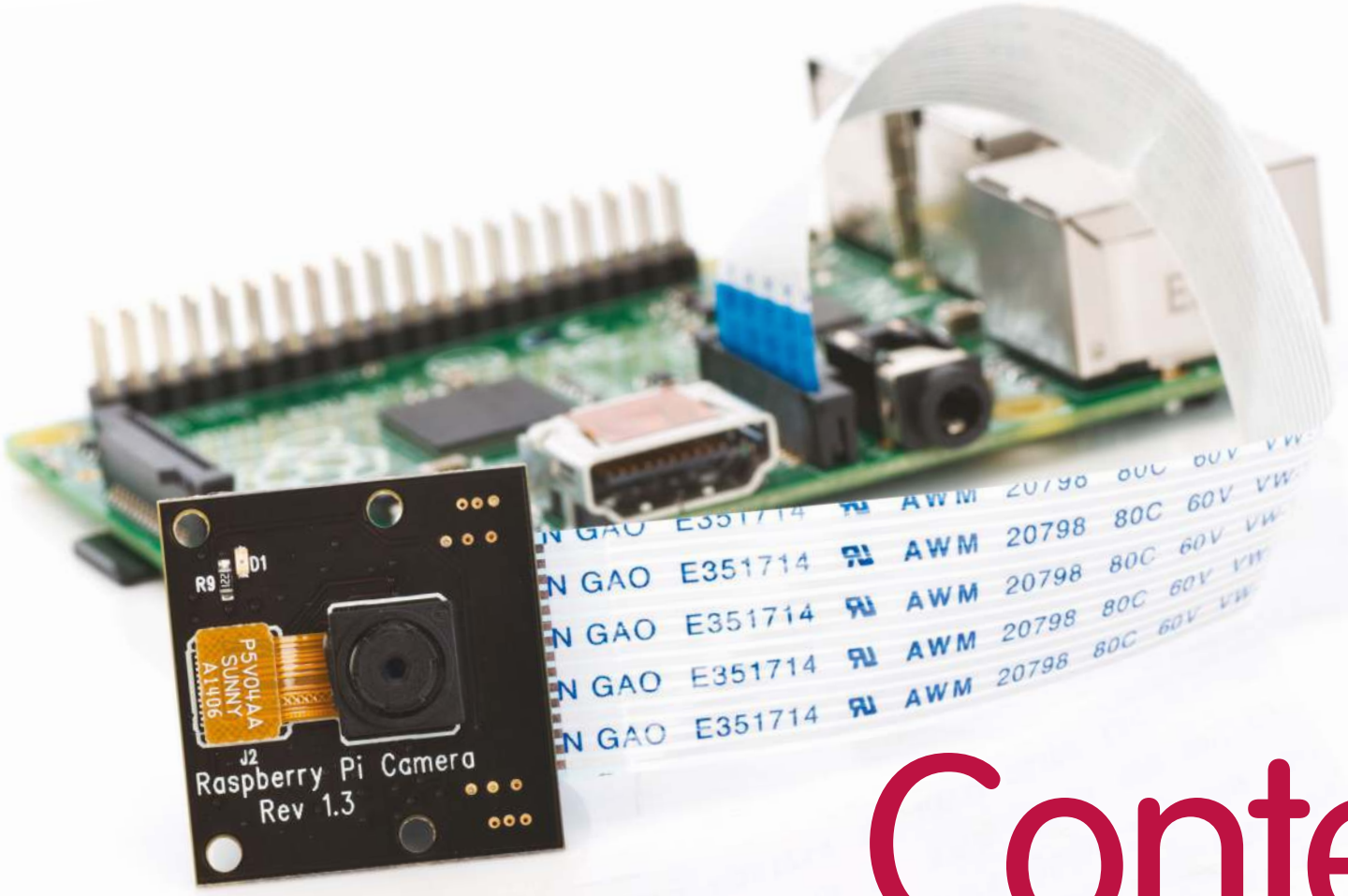
Learn to make and code gadgets with Raspberry Pi

From the makers of

LinuxUser & Developer

Join the conversation at...

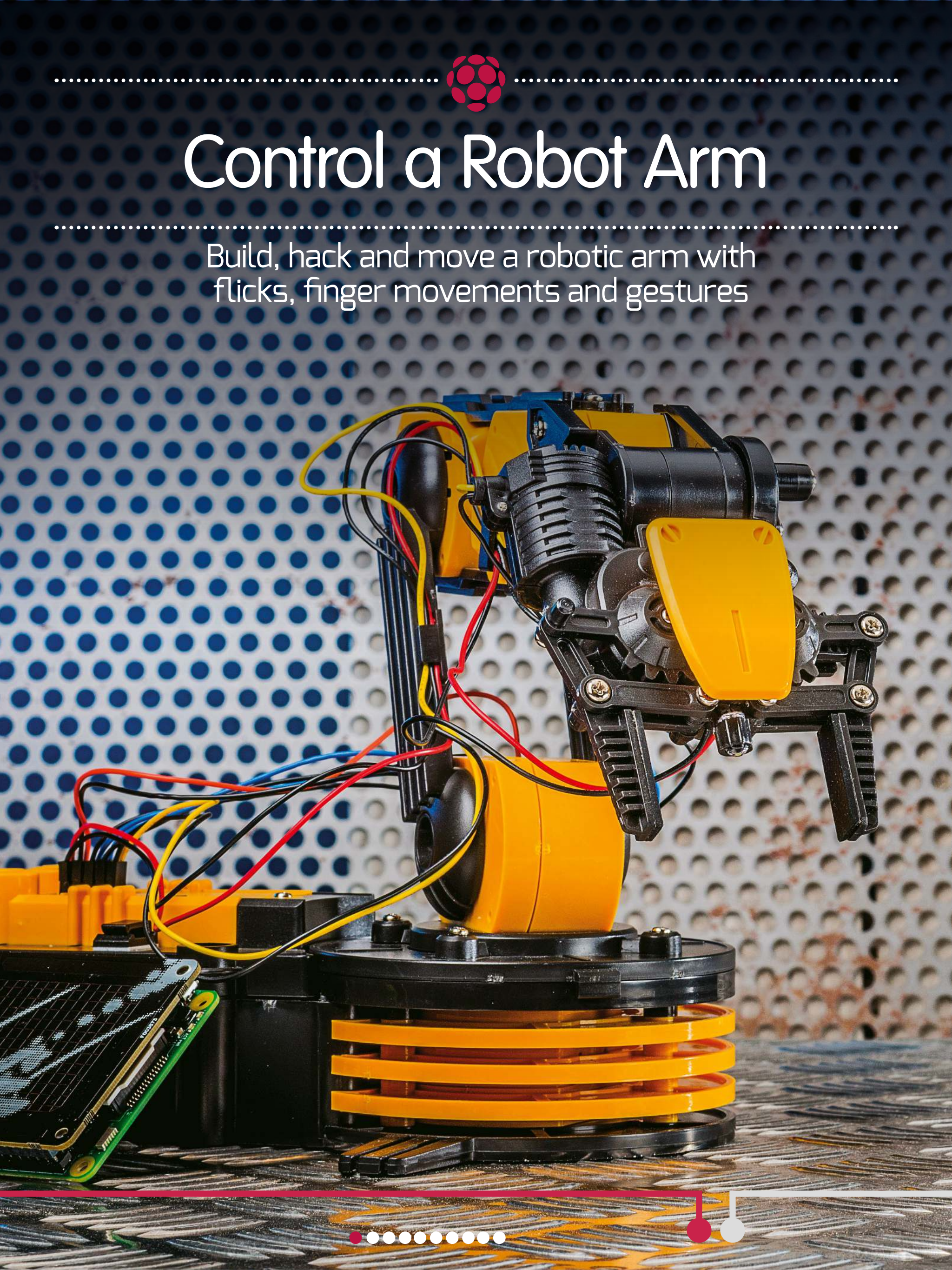
@linuxusermag | Linux User & Developer | linuxuser@futurenet.com



Contents

Control a Robot Arm

Build, hack and move a robotic arm with flicks, finger movements and gestures

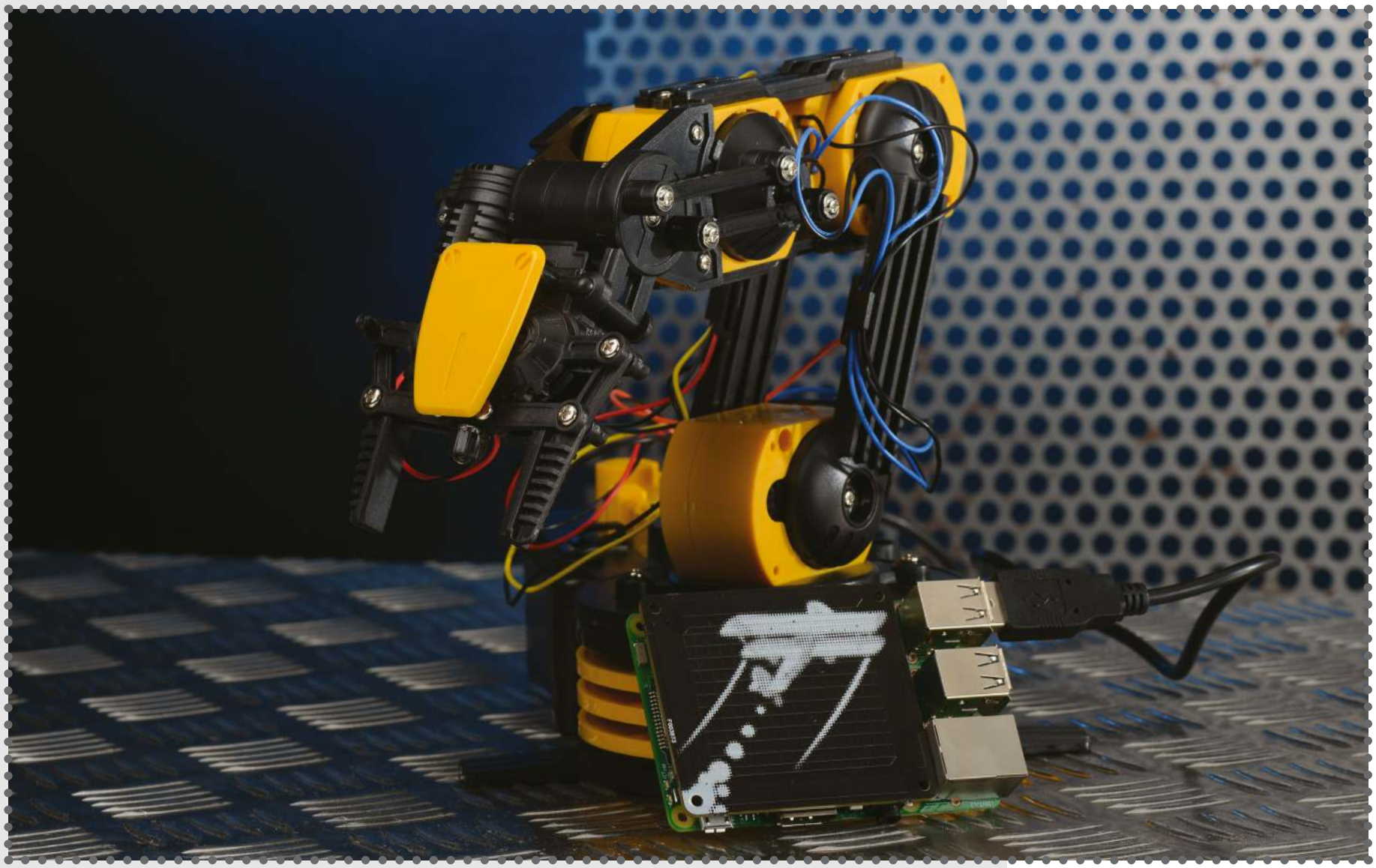


This tutorial combines the Maplin Robotic Arm and Pimoroni's Skywriter HAT so you can take control of the robot with just the touch of your fingertips. The arm can move through three points of articulation and ends in a clamp to give you maximum flexibility with movement. The arm also moves through 120 degrees in the wrist, 300 degrees at the elbow, 180 degrees at the base in the vertical and 270 degrees in the horizontal. It is easy to assemble and is a great beginners' robot. In this tutorial you will first install the Python modules to enable your Raspberry Pi to interact with the USB port, and then learn how to use Python code to send commands to and receive them from the arm. Next we'll write a simple program to control the arm; this is actually great fun and you can start to practice picking up objects and then trying to move them to another location.

The Skywriter is an electrical near-field 3D sensing board that recognises a number of gestures including taps, double taps and flicks. These gestures can be made at a specific location on the board, either north, south, east, west or in the centre, and then programmed to respond in a particular way. The second half of the tutorial shows you how to install and set up the Skywriter and code some of the gestures. Finally, both the arm and the Skywriter code are combined to create a program where you assign a particular arm movement to a particular gesture, creating the controls at your fingertips. For example, a tap on the north of the board could move the arm up, a tap at the bottom (the south position) could move it downwards. Watch this video to see the project in action: **https://www.youtube.com/watch?v=SVDXbcSi08I**

THE PROJECT ESSENTIALS

Maplin Robotic Arm
Pimoroni Skywriter
Raspberry Pi 2 or 3



01 Build your Robot Arm

The Robotic Arm comes in kit form with a detailed set of instructions covering how to build the parts and combine them into the final arm. This will take you around three or so hours which is perfect for a rainy day. The arm kit includes all the required parts except for the batteries. The trickiest part is to ensure that the wiring is the correct way round. This will not damage the arm but it will result in the arm moving in the opposite direction to the programmed direction.

02 Update your Pi

After building the arm, plug in your Raspberry Pi and boot it up. At this point do not plug in the arm. First update the software and OS. Ensure that your Pi is connected to the Internet, open the

LX terminal window and then type the following,

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

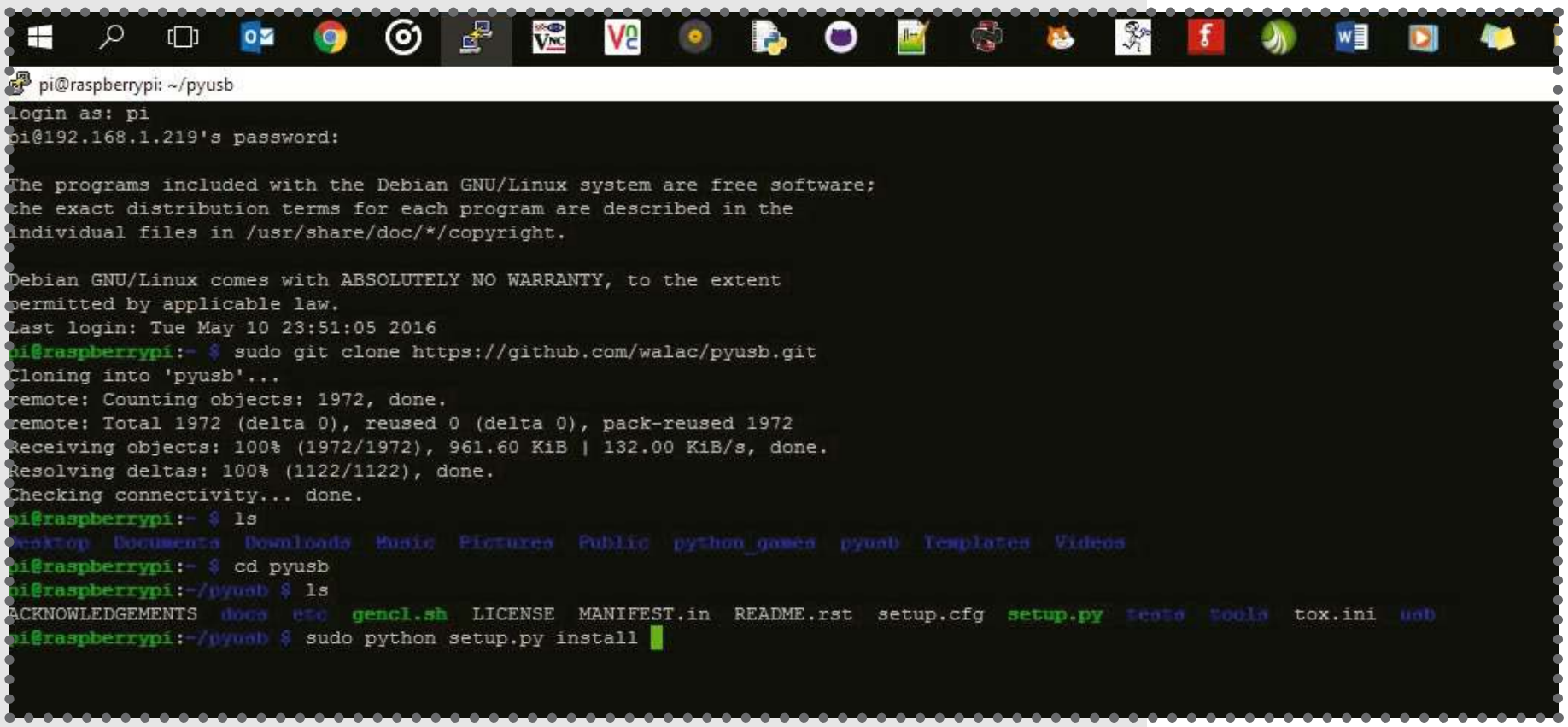
Wait for the software to download and install, then reboot, typing sudo reboot.

03 Install the PyUSB software

To use the Python programming language to interact with your robot arm, first install the PyUSB module. It provides a connection with Python and easy access to your Raspberry Pi's Universal Serial Bus (USB) where the arm is plugged in. In the terminal window, type:

```
sudo git clone https://github.com/walac/pyusb.
  git
```

Once installed move to the pyusb folder, type cd pyusb, then type the code sudo python setup.py install. This will install the required module. Finally, reboot your Raspberry Pi with sudo reboot.



04 Attach the robotic arm to the Pi

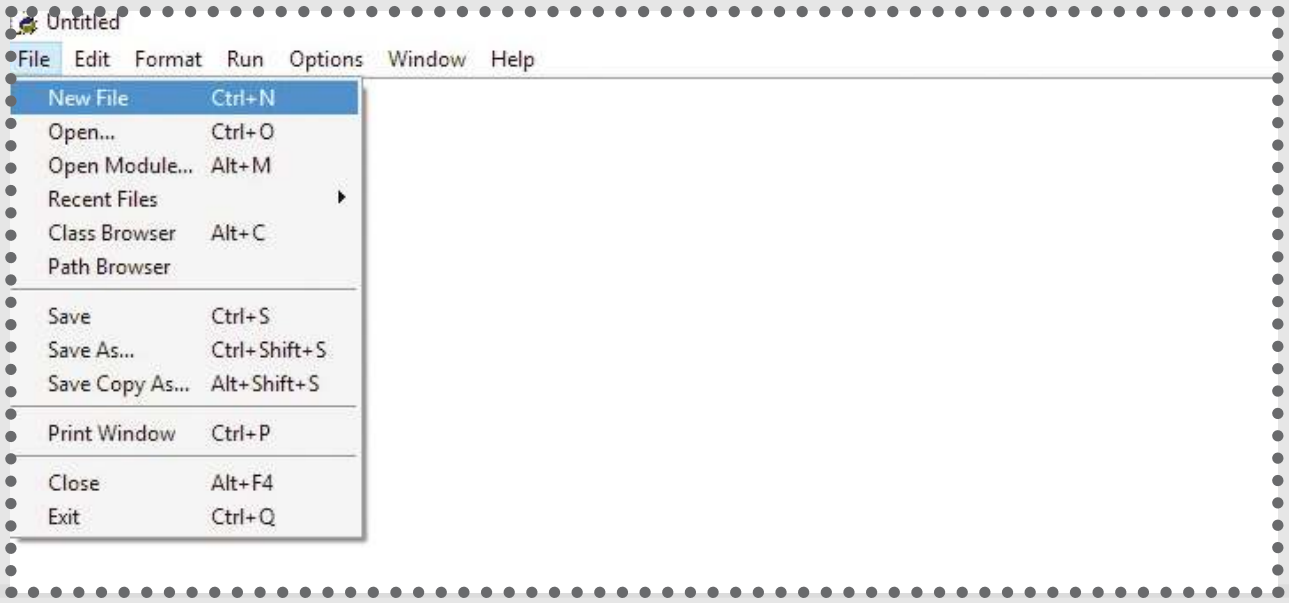
Now that you have installed the required software to program the arm, plug it in. To do this simply take the USB lead from the arm and slot it into one of the available USB sockets on your Raspberry Pi. Turn the arm on using the power switch located on the base of the arm. You can test that the connection is live and the arm is recognised by opening the LX Terminal and typing sudo lssub. This will return a list of the hardware connected to the USB ports.

05 Take control of the arm

The arm is controlled by sending a line of code via the USB to the arm. This code includes the duration time of each movement as well as the coordinates to identify which motors to turn and on or off and in which directions to turn them. The actual codes to move the arm are fairly verbose, however Python simplifies them into manageable single lines of code.

06 Get a new LIRC file

From the menu of the taskbar select and open the LX Terminal, double click the logo. Administrative privileges are required to access the arm via Python, so open Python as a sudo user. In the LX Terminal type sudo idle and press Return. When IDLE has loaded, open a new window, click File and then select New Window.



Experiment with the gestures

The Skywriter boasts a number of gestures that you can use in your own program. For example, the board can be used to detect you making a circular gesture, referred to as an air wheel. The Skywriter is also perfect for controlling additional projects, like changing the volume on your music player, or using the gestures to create music or play samples. Check out the full code for these examples and more at **https://github.com/pimoroni/skywriter-hat/tree/master/examples**

07 Import the required libraries

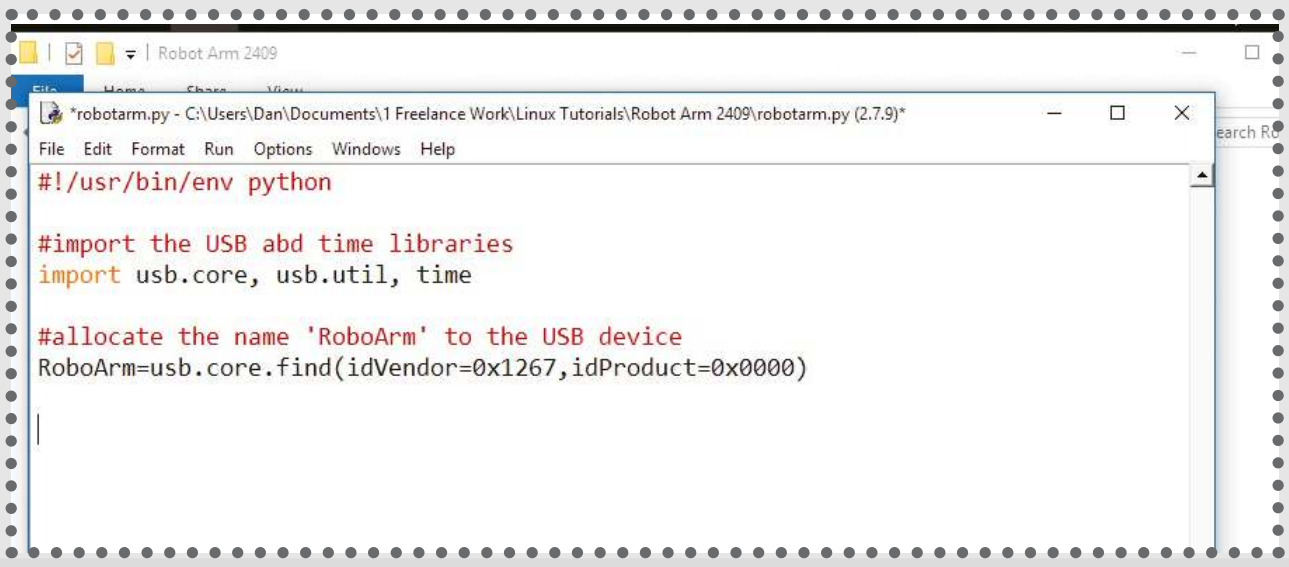
At the top of the window, create your program. Import the time, core usb and usb utility libraries, line 1. These enable you to send commands to the USB port and control the arm. USB is a complex protocol, but PyUSB has good defaults for most common configurations, minimising the need for verbose amounts of code. The time library permits you to add short delays between each movement of the Arm. Add the line of code as below.

```
import usb.core, usb.util, time
```

08 Name the USB device

Now use Python to find the USB arm. On the next line down, create a variable called RoboArm to store the information about the location of the robotic arm – this is basically which USB port it is plugged into. Now name the device that is plugged into the Raspberry Pi USB port. To do this, enter the ID Vendor number and the product ID details to search for the arm. If you are using a different arm or model number, you will need to locate these two details and replace them in the line of code.

```
RoboArm=usb.core.find(idVendor=0x1267,idProduct=0x0000)
```



09 Find the robotic arm

The next step is to search for the robotic arm to ensure that it is plugged into the USB port and also turned on. This uses a simple conditional, an if statement, to make this check. If the arm is not found, then an error message is returned to the Python window to let you know to plug it in, turn it on, or both! Note the second line of the code is indented.

```
  if RoboArm is None:
     raise ValueError("Arm not found")
```

10 Create a function, part 1

You need a small delay between each movement of the arm. Create a new variable called Duration and assign a value of 1, line 1. On the next line down, create a function that includes the duration value you just created, and also a code to stop the arm moving. On line 3, add the code to transfer the movement commands to the USB port and relay them to the robotic arm.

```
  Duration = 1

  def MoveArm(Duration, ArmCmd):
     #start the movement
```

Robot arm movements

Use the list below to reference the robot's movementds. The text after each # is a comment and used to identify the command.

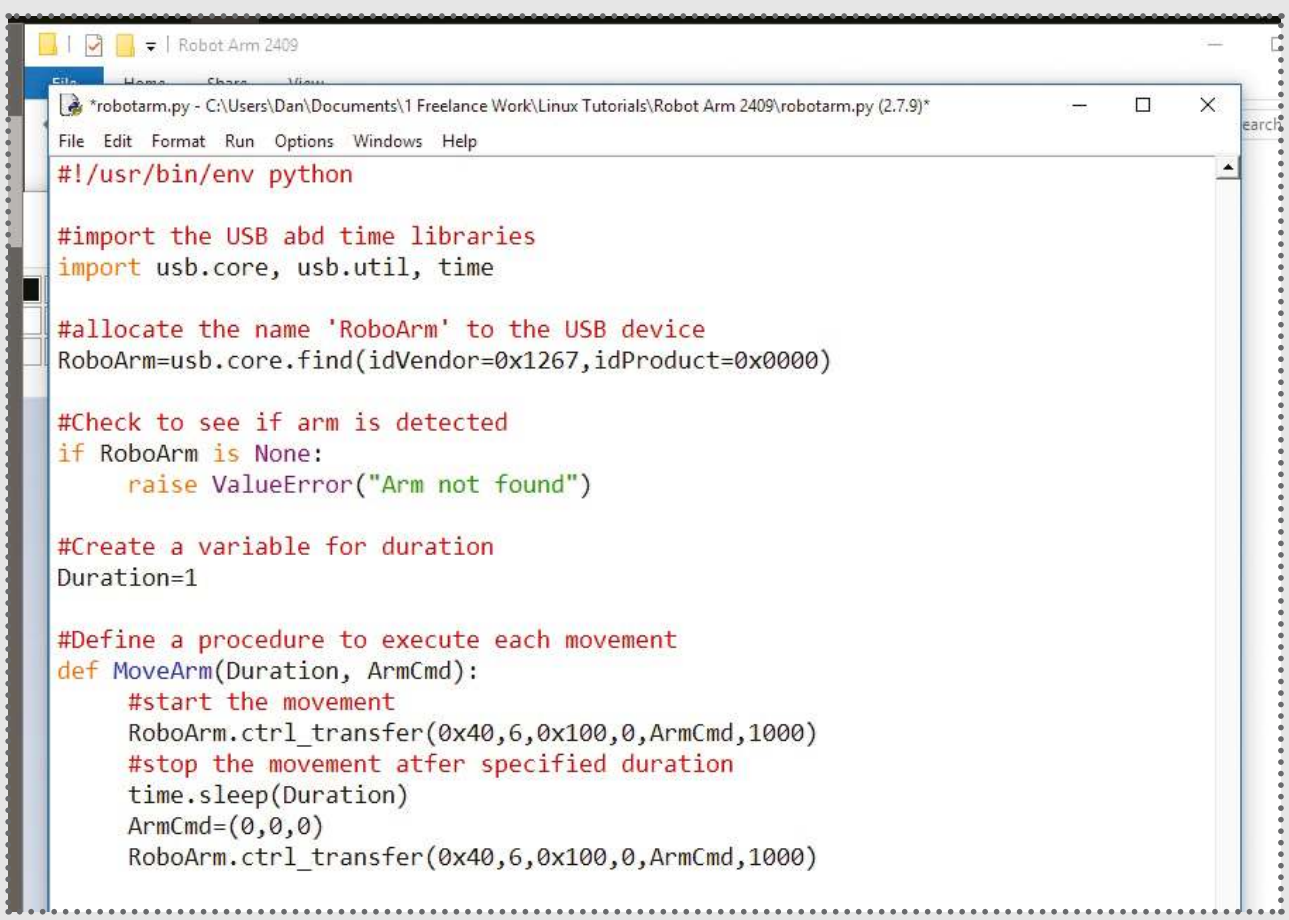
```
MoveArm(1,[0,2,0])
#rotate base clockwise
MoveArm(1,[64,0,0])
#shoulder up
MoveArm(1,[128,0,0])
#shoulder down
MoveArm(1,[16,0,0])
#elbow up
MoveArm(1,[32,0,0])
#elbow down
MoveArm(1,[4,0,0])
#wrist up
MoveArm(1,[8,0,0])
#wrist down
MoveArm(1,[2,0,0])
#grip open
MoveArm(1,[1,0,0])
#grip closed
MoveArm(1,[0,0,1])
#light on
MoveArm(1,[0,0,0])
#light off
```

```
    RoboArm.ctrl_transfer(0x40,6,0x100,0,ArmC
md,1000)
```

11 Create a function, part 2

Add a small duration time to set a limit for the action. In this example the duration is set to 1, meaning that each movement or action will happen once. You can experiment with the duration time by changing the value you set in Step 10. After each of the movements occur then they need to stop. Use the code ArmCmd=(0,0,0) line two, to prepare a code to stop the motors turning and end the movement.

```
  #stop the movement after specified duration
    time.sleep(Duration)
    ArmCmd=(0,0,0)
```



12 Create a function, part 3

The last part of the function is to send the stop command to the arm, line 1. This connects to the arm

Create a UI

It is possible to combine the IR board, the LIRC program and a web interface, which would open up many possibilities for projects. Combining the GPIO pins with a web server means that you can create a user interface that can be used to control your devices. Change the channel from your laptop or phone, turn the volume up or down or turn the TV off. A starter project can be found at **bit.ly/1O4CaMU**.

and then transfers to the ArmCmd=(0,0,0) code that you set up in the previous step. Note that it is stored in a variable called ArmCmd which means that you can change the command that is transferred to the arm. This line is also indented.

```
    RoboArm.ctrl_transfer(0x40,6,0x100,0,ArmC
md,1000)
```

13 Movement and action code

Once you have completed the function, you can now use it to control the arm. Each movement or action has a unique set of digits that identify how long the movement last for, which motors need to be turned on or off and in which direction they need to turn. Each command begins MoveArm, then the first number is the duration the movement lasts for. For example, '1' is basically, do this instruction or movement for one second. This is then followed by the required motor information to turn it: MoveArm(1,[0,1,0]) #rotate base anti-clockwise.

14 Turn a light on

On the next line down, underneath the function, add the code to turn the light on. Type the command MoveArm(1,[0,0,1]) #light on. The #light on is a comment and is not required to control the arm. It is used to identify to the user what the code does: it turns the light on. This is useful when you have several commands in the program.

15 Run the program

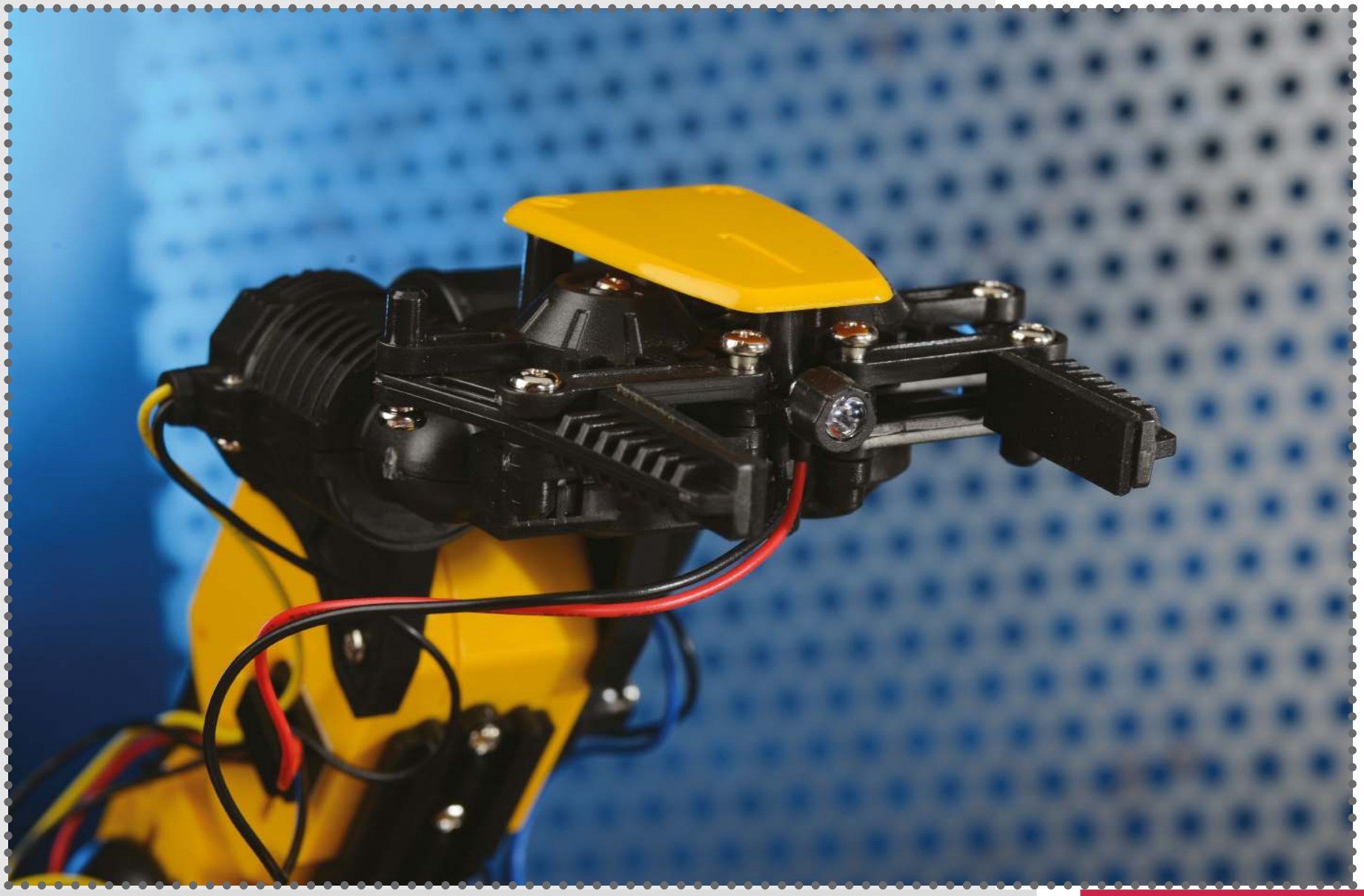
Now you have a complete program that will connect to the USB port, search for the robotic arm, prepare a

"Sometimes you may not find a compatible lirc.conf file and instead you have to create one yourself"

function and then transfer an instruction that controls the arm, turning the light on. Save the program into your home folder. An easy way to do this is to press F5 on the keyboard, name and save the file, press Enter and then it will run. Look at the light on the arm, it will be on!

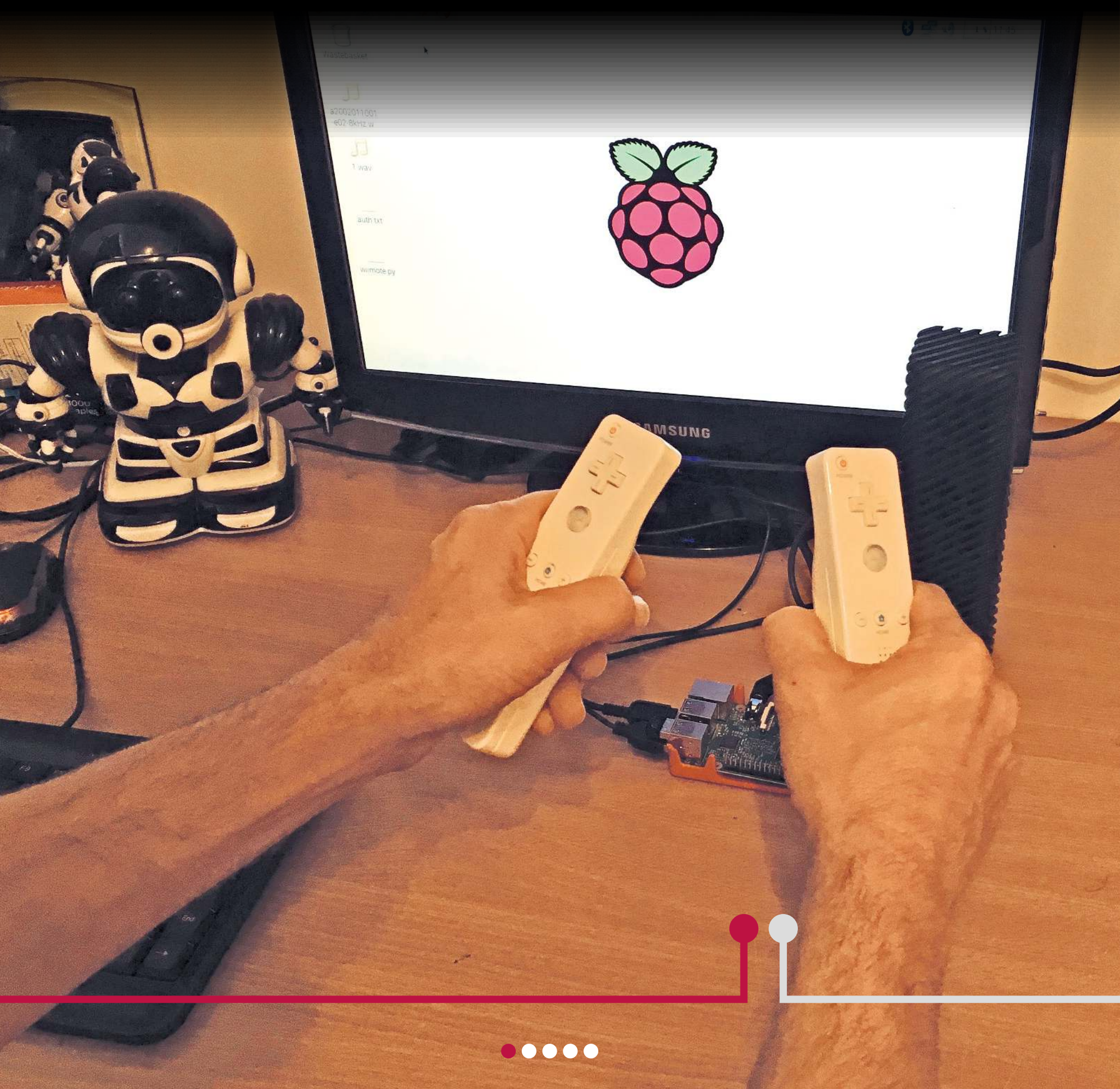
16 Turn the light off

The light will now stay on unless you code it to turn off. If you add this line straight after the 'turn on' command, then the light will switch on and off so fast that you will not see it. On the next line down, add a short delay of two seconds, line one, and then turn the light off, line 2. Save and run your program using the F5 button. Check out the full code listing to see the code for all the available movements. You can use these code lines to build up movements and experiment with the arm's functions.



Raspberry Pi air drum kit

David Pride's air drum kit turns the Raspberry Pi into a musical maestro



Where did the original idea for the drum kit stem from?

What's always interesting to me is where we find our sources of inspiration. These can be a person, a book, a tweet, a website – anything at all. A lot of my project ideas start when I find something at the local car boot sale.

This time what I found was an Air Play 'Air Drum' – being offered for the grand sum of £1! How could I possibly refuse? So I took it home and had a quick play and to be honest, while the concept is great, the actual functionality was a bit limited, the sound quality was rubbish and it was also a bit suspect as to what sound played with what movements.

How was the build process? Did you encounter any issues?

This got me to thinking whether I could make something more effective using a Raspberry Pi. We've been playing around with Wii controllers at both Cotswold Raspberry Jam and Cheltenham Hackspace recently. We've built several mini-bots for a bot-versus-bot challenge known as 'Pi Noon'. Neil Caucutt from Cheltenham Hackspace has done an amazing job designing the chassis for these bots. They use the excellent Python cwiid library that lets you use Wii controllers with the Raspberry Pi. I'd only managed to ever get one controller working with a single Pi before so the first challenge was to get a pair of controllers working as the 'sticks'. Once I'd identified that by using the MAC address of each controller, multiple

David Pride is a Raspberry Pi devotee who has played a major role in Pi-centric events throughout the UK.

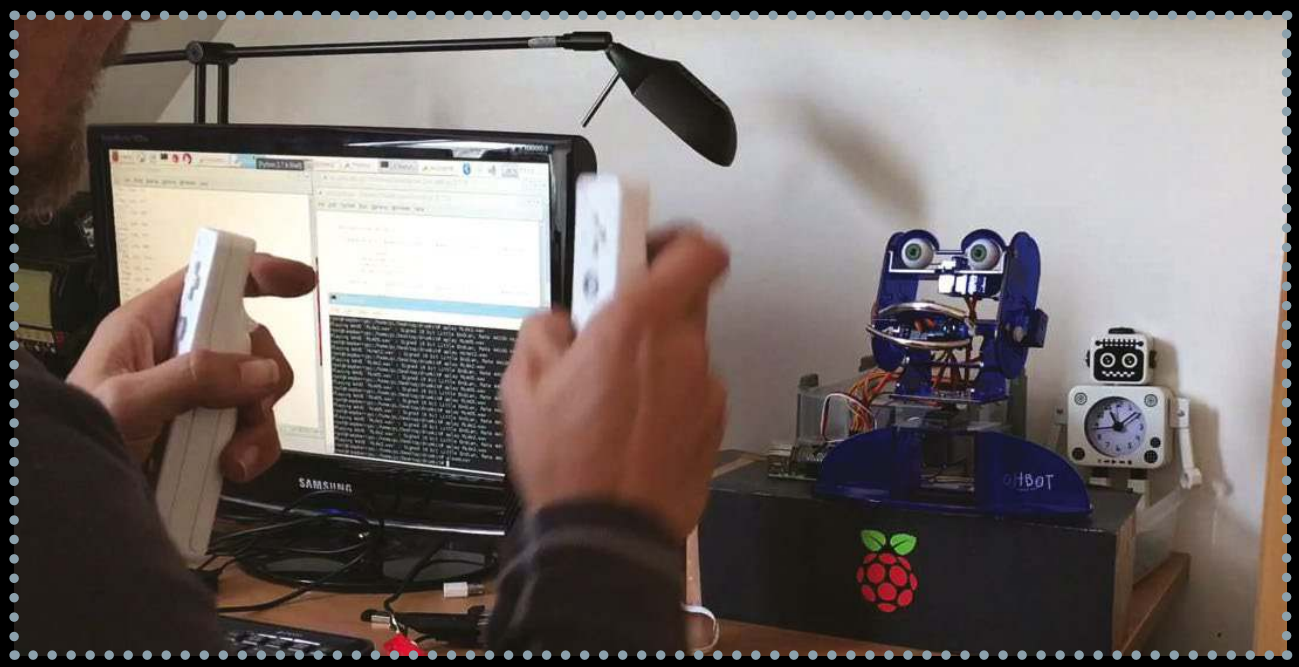
controllers can be 'read' by a single Pi this gave me the ability to set up two controllers as a pair of drum sticks.

How are controller movements actually tracked?

I found a bunch of open source drum samples that were available as .wav files – there are literally thousands of these out there to choose from. I then wrote a small TkInter app that displayed the position of the controller to give me an idea of the data that was being produced. Interestingly the position and accelerometer data from the Wii controller is all wrapped up in a single xyz Python tuple. This caused some confusion initially as if you move slowly from point A to point B this produces a very different reading than if the same movement is done rapidly. After playing around for a while (quite a long while!), I managed to map four distinct movements to four different drum sounds.

Were there any limits to the sounds you can implement?

I initially wanted to get three sounds on each controller, but the movement scale was a bit too tight to do it successfully every time and two sounds



THE PROJECT ESSENTIALS

- Raspberry Pi 3
- Nintendo Wii controllers
- Silverlit Air Drum Kit
- Python cwiid library
- TkInter
- Wooden buttons
- Open-source drum samples

often overlapped. So, I am using the trigger button combined with the movement for one of the sounds on each controller. This gives six different drum sounds, three per controller, that can be played easily without them overlapping.

Did you find the Pi a good board?
I found the response time to be very acceptable on a Pi 3, I posted the code to Github and others have also been having fun with it. I've seen one version that uses my controller code and PyGame to play the sound files; this seems to work better on older versions of the Pi.

Is there any way you see this project being expanded on? Integrating more sounds perhaps?
In regards to what else could be done, I am interested in seeing if the actual mechanics from the Wii controllers could be mounted in a pair of gloves; this could be a really interesting experiment. Additionally, I'd like to configure a more refined output that can

Like it?

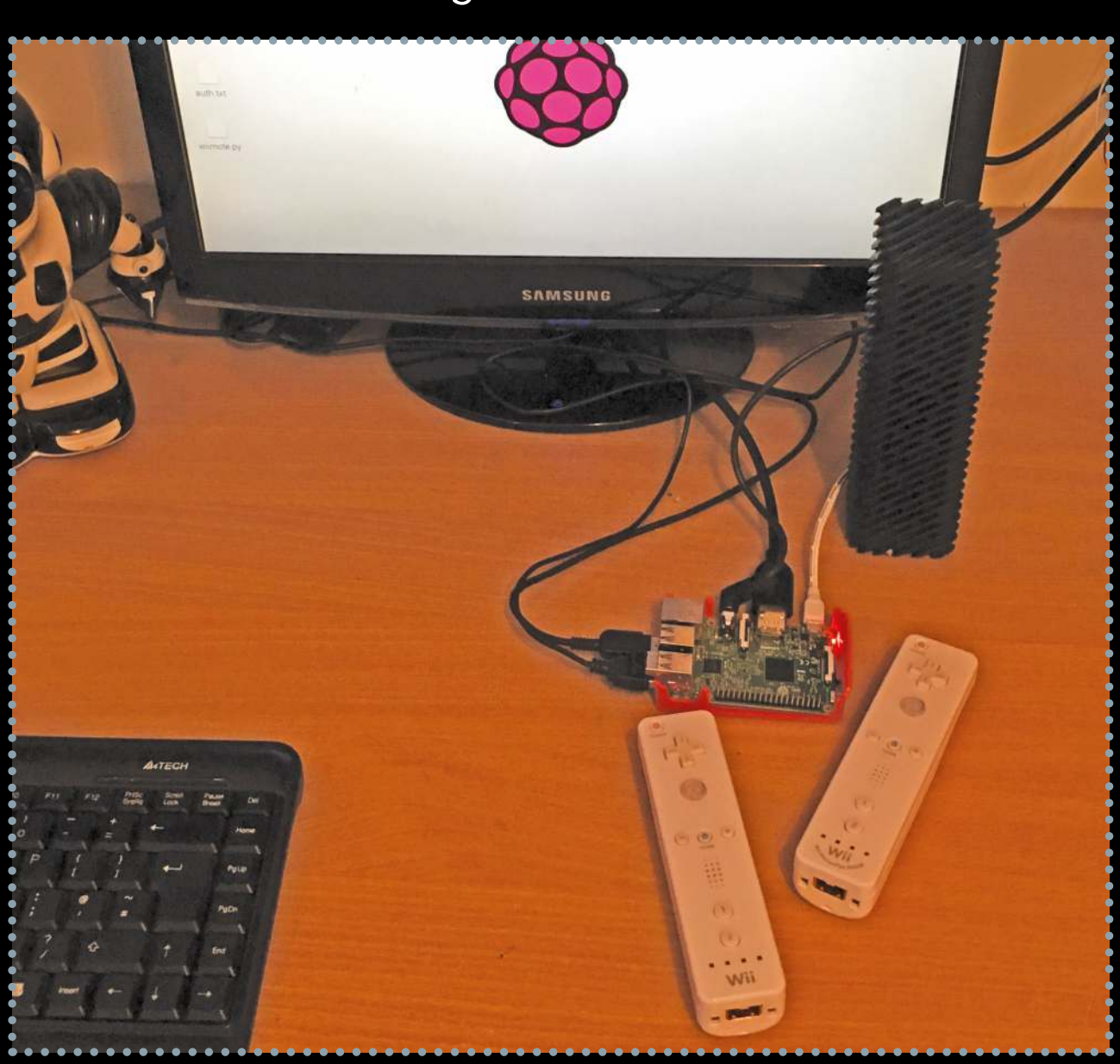
David has been massively involved with the Pi community for a number of years, and we've featured several of his projects previously. We highly recommend you check out his Connect 4 robot, which was another novel way into integrating the Raspberry Pi into a different type of project: **http://bit.ly/2fbsSnW**

Left The project started from David's purchase of a Silverlit Air Drum Kit, which was heavily modified for compatibility with the Raspberry Pi unit

generate MIDI signals rather than just playing stock sound files, this would really open up a whole range of different possibilities.

Many of our readers will know about your Pi exploits. What's next for you? Any big projects in the pipeline?

In regards to what comes next, I am very fortunate in that I've just got my hands on a 3D printer so have been having a lot of fun experimenting with that. I've designed a LEGO-compatible case for the tiny Raspberry Pi Zero, which is proving extremely popular. I've been selected to take part in Pi Wars, the Raspberry Pi robotics competition that takes place in April 2017 so I'll be doing a lot of preparation for that event too in the coming months.



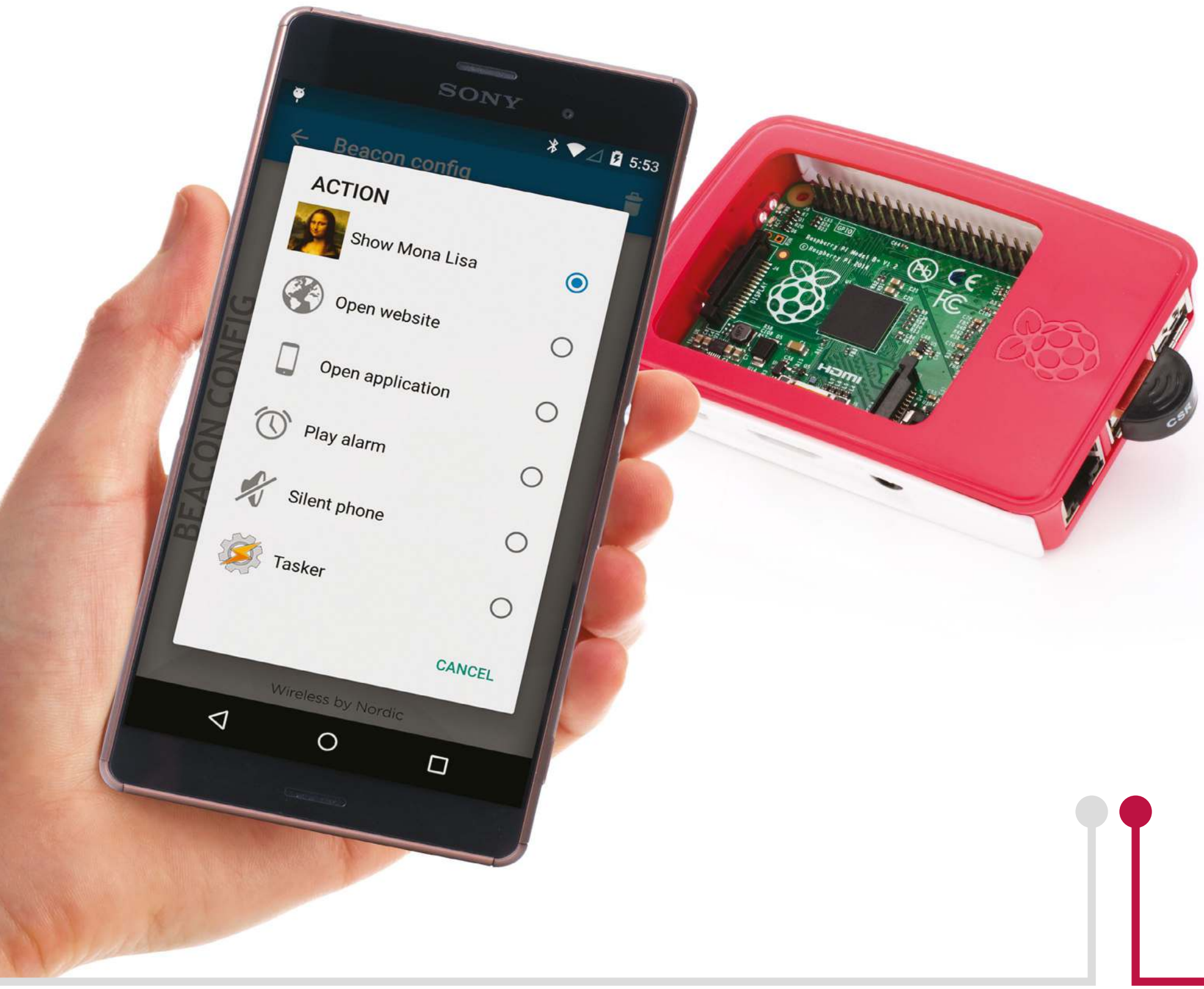
Further reading

While the air drum kit may be a niche project to undertake, David does explain that this is very much a beginner-friendly project to get started with. A visual look at the project can be found over at: **http://bit.ly/2gnkeEB**, while all the necessary code can be yours from his official GitHub listing: **http://bit.ly/2gnll7b**

Left Both motion and speed are tracked directly by the Pi unit, which helps to then trigger the drum and cymbal sounds achieved by the controllers

Use your Raspberry Pi to find and track your phone

Create a program that locates Bluetooth devices and responds to them



The Raspberry Pi model 3 saw the introduction of embedded Wi-Fi and Bluetooth capabilities. This now makes it even easier to interact with Bluetooth-enabled devices such as mobile phones, tablets and speakers. The Python programming language supports a range of libraries that enable you to interact, monitor and control various elements of a Bluetooth device. This tutorial combines the Pi's Bluetooth hardware with Python code to create three simple but useful programs. First, build a short program to search for Bluetooth-enabled devices and return the 12-part address of each device. Once you have obtained the addresses, you can then scan and find Bluetooth services that are available on that particular device. Finally, use the Bluetooth address and some conditions to check which devices are present in a building and in turn, which people are present or absent in a building, responding with a desired action – it's a kind of automated Bluetooth checking-in system.

THE PROJECT ESSENTIALS

Raspberry Pi 3

Bluetooth USB dongle (if using an older Pi model)

01 Using Bluetooth

Bluetooth is a wireless standard for exchanging data between devices over short distances of between one and ten metres. The current version, Bluetooth v5, was notified in June 2016 and has an increased range of over 200 metres. It will be released in 2017 and will probably be a staple of many IoT devices and applications. Bluetooth uses the standard IEEE 802.11, which is the same standard as Wi-Fi. They both have similarities such as setting up a connection, transferring and receiving files and streaming audio and media content. If you have a Pi 2 or lower, you can still create and use these programs by using a Bluetooth USB dongle.

02 Install the required libraries

Although the Raspberry Pi OS image comes with a Bluetooth library for interfacing with devices, for this tutorial you will want to control the interface with Python code. Load up your Pi and open the LX Terminal window. Check for and update/upgrade the OS, typing in lines 1 and 2. Then install the Python development tools, line 3. Finally install two further Bluetooth development libraries. Once they have completed, restart your Pi by typing
sudo halt.

```
  sudo apt-get update
  sudo apt-get upgrade
  sudo apt-get install python-pip python-dev ipython
  sudo apt-get install bluetooth libbluetooth-dev
  sudo pip install pybluez
```

03 Load Python

Load the LX Terminal and type sudo idle, this will open the Python editor with super-user privileges, which will give you access to the USB hardware via Python code. Start a new Window and import the first Bluetooth module, line 1. Then add a short message to inform the user that the program is searching for nearby devices.

```
  import Bluetooth
  print("Searching for device...")
```

04 Search for the names of the devices

The next line of your program searches for the names of the Bluetooth enabled devices. Each device must have Bluetooth

Is your dongle working?

If you are using a USB Bluetooth Dongle then you can check that it is working by plugging it in, then restarting your Raspberry Pi by typing the command sudo reboot. When reloaded, check that the Dongle has been recognised; load the LX Terminal window and type lsusb. This will list all the connected USB devices. You can also type hcitool dev to identify the dongle and return its USB address.

enabled and be discoverable in order to be found. On the next line down, create a variable called **nearby_devices** to store the names and use the code **bluetooth.discover_devices** to look up the names of the devices.

```
nearby_devices = bluetooth.discover_devices(lookup_names = True)
```

05 Print the total number of devices found

Each of the names of any discoverable devices are now stored in a variable. Use the code **len(nearby_devices)** to return the number of items stored in the variable, this is the number of Bluetooth devices the program has found. Then print out the total number of devices. Add the following code after the previous line of your program.

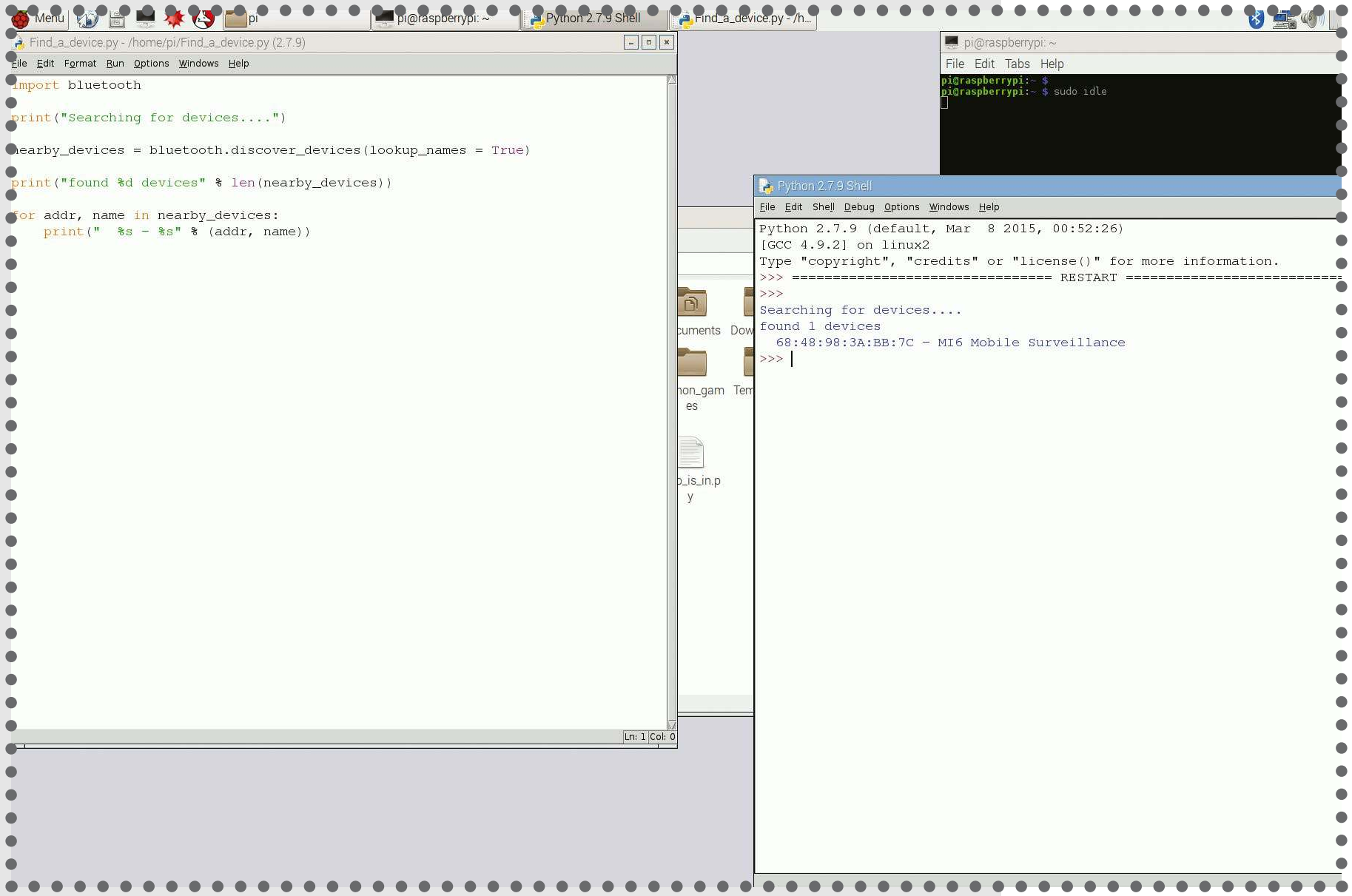
```
print("found %d devices" % len(nearby_devices))
```

06 The Bluetooth address (BD_ADDR)

Each Bluetooth-enabled device has a Bluetooth address that is a combination of 12 alphanumeric characters; for example, 69:58:78:3A:CB:7F. The addresses are hexadecimal, which means they can contain numbers from 0 to 9 and letters from A to F. Most devices manufactures will include the address on a sticker attached to the hardware or within the user manual.

07 Print the name and address

For each of the devices that the program has located, (each of the items in the list), print out the address of the device and also the name of the device. This information is used in the

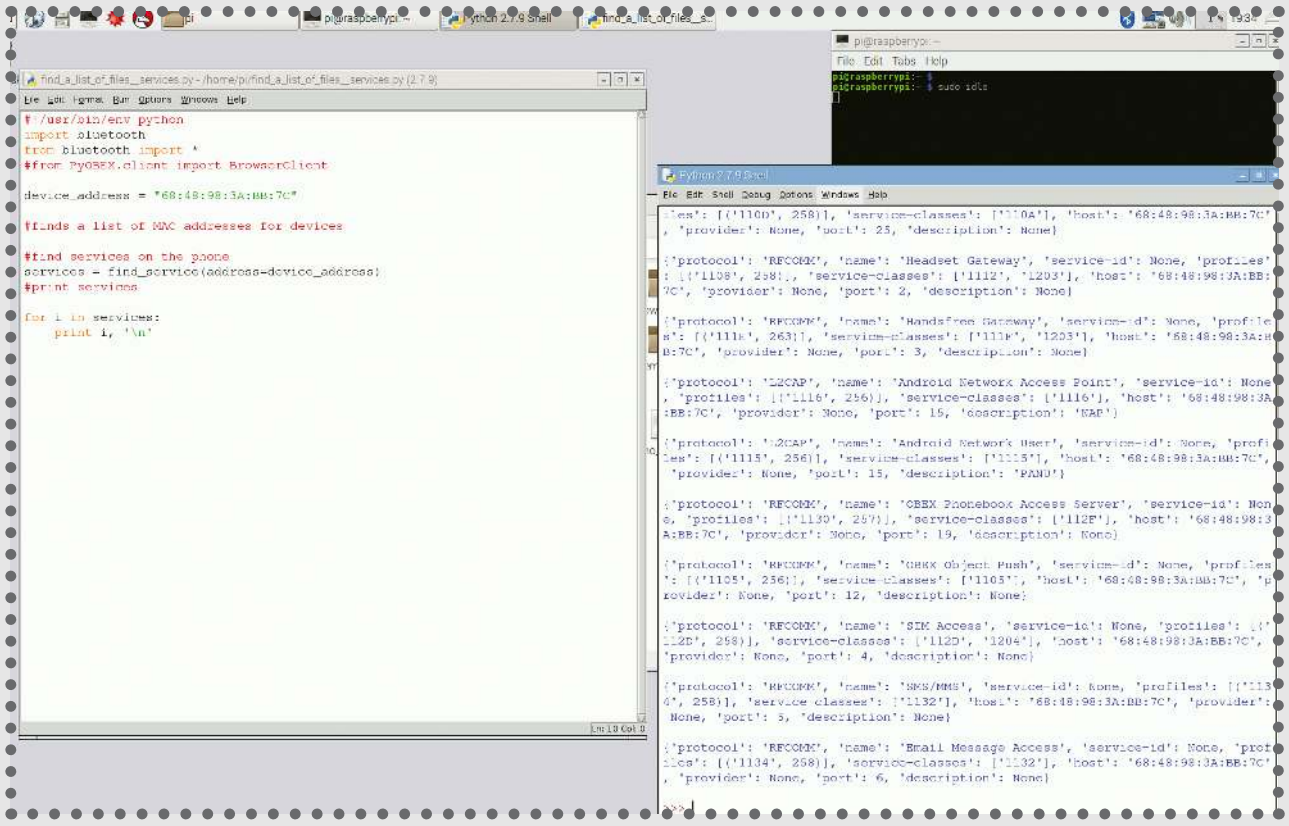


next sections to find out what available services a device has and also to add an action when a device is found. Save the program and then run it, ensuring that the any devices to be found are in Discovery mode. You will be presented with a list of the devices that includes the name and address of each.

```
  for addr, name in nearby_devices:
    print("  %s - %s" % (addr, name))
```

08 Find the available services on a Bluetooth device

Run the previous program and write down the Bluetooth address of the device. Start a new Python program and save it. Next open up a new Python window and start a new program. Input the two required libraries, lines 1 and 2.



09 Set up the address to find

On the next line down, enter the address of the device that you want to find the services on. Use the previous program or look at the sticker to obtain the address. Next, create a variable called 'device_address' to store the address. Use the following code and replace the example address with the Bluetooth address of your device.

```
device_address = "69:58:78:3A:CB:7F" # enter address of device
```

10 Find a service

On the next line down, add the code to find the services that your device supports. Create a new variable called services, which will store a list of the services. Use the code, find_services, followed by the Bluetooth address of your enabled device to search through a list of available services and store each of them in the 'services' variable.

```
services = find_service(address=device_address)
```

11 Print out each service

The last step of the program is to print out each of the services. These are stored in a list and therefore need to be printed out one line at a time. Create a loop using the code, for i in services, line 1. This loop will check for each of the individual items in the list. It will then print each of the items in the list, each Bluetooth service, line 2. Use the code '\n' to print each service onto a new line.

```
for i in services:
    print i, '\n'
```

12 What are the services?

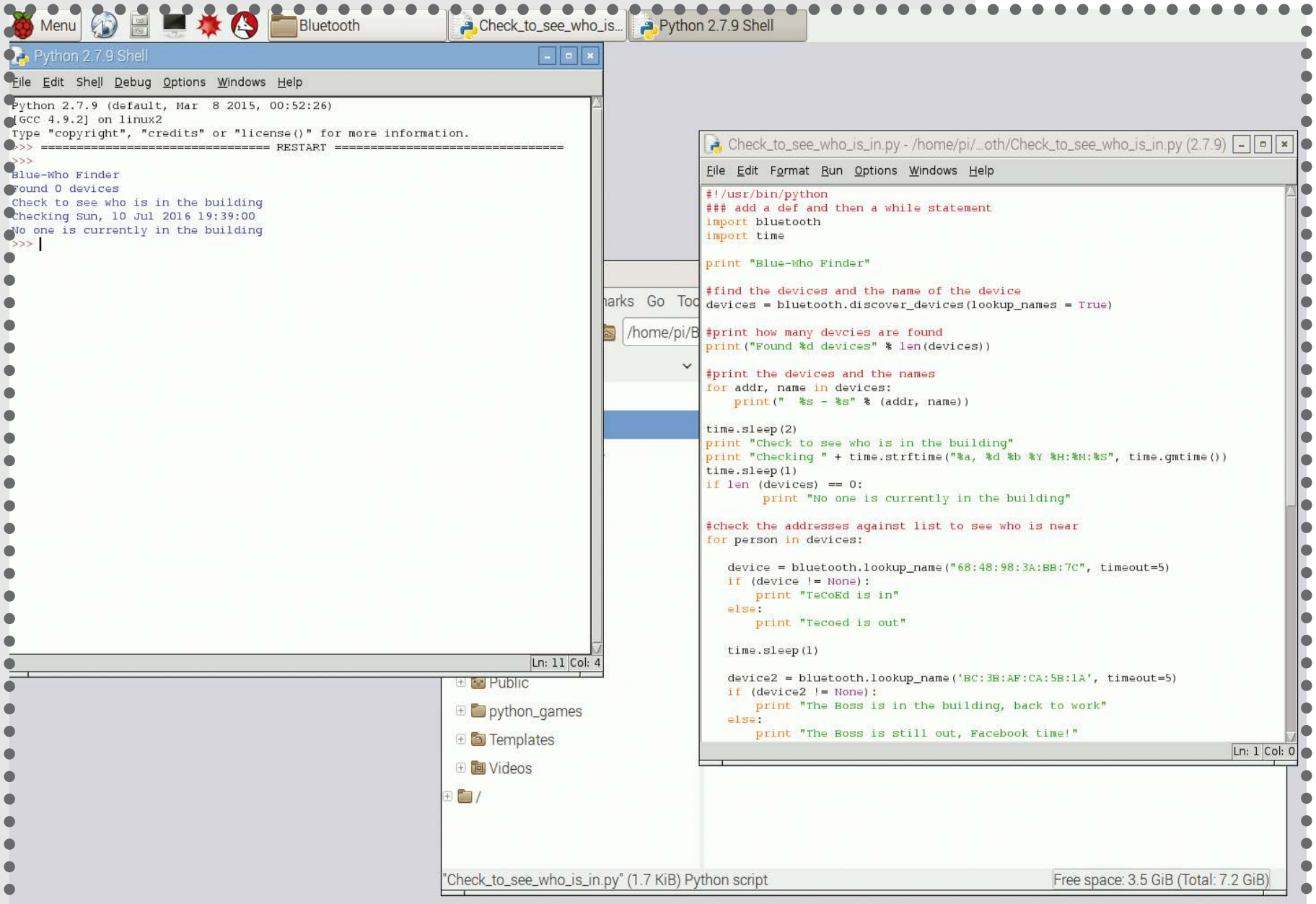
Save and run your program and you will be presented with a long list of services, especially if you are using a modern device. Using these services with Python is more complex and requires several other lines of code. However, you can potentially transfer and receive files, stream music and even shut the device down. There are more details and commentary on the Blueman Github page, **https://github.com/blueman-project/blueman**. Remember though that these tools are purely for personal use.

13 Find a device and a person

In Step 7 you used a short program to discover the Bluetooth-enabled device and check and return the address of the device. Now use the bluetooth.lookup_name code line to search for a particular device and return whether it is found or not. If it is found then the device is present and if not, then we can assume the device is not. However, remember that the Bluetooth may be turned off. In your Python program add the line to locate the device, replacing the example address with the address of the one that you want to locate.

Is your Bluetooth Dongle compatible?

If you have an older Raspberry Pi model 1, 2 or the Pi Zero then Bluetooth capability is not included. However, you can buy a USB Bluetooth dongle and attach it via one of the USB ports to add capability. Not all Bluetooth dongles are compatible so there is a developing list of the tried and tested ones available. Check here before you purchase one: **http://elinux.org/RPi_USB_Bluetooth_adapters**




```
device = bluetooth.lookup_
name("33:38:33:6A:BQ:7C", timeout=5)
```

14 Respond if the device is found

Once a device has been searched for, check to see if it is present and responds, line 1. Use an IF statement to see if the device is not found. This uses the symbol '!=', which means 'is not'. However, the code is checking if it is not 'None' – in other words the device is found. If it finds the named device then print out a message, line 2. If it does not find the device, line 3, then print out a message to notify the user, line 4. Add these lines of code underneath the previous line. Ensuring that your Bluetooth is enabled, save and run the program to find your device.

```
  if (device != None):
    print “TeCoEd is in”
  else:
    print “Tecoed is out”
```

15 Find a different device

To find other devices and respond with an action, simply use the same sequence of code but first create a different variable to store the response in. Add the code on the next line down and remember to de-indent it. Rename the variable, for example call it device_one and edit the address to match that of the second device.

```
Device_one = bluetooth.lookup_
name(“44:67:73:6T:BR:7A”, timeout=5)
```

16 The next device is found

As before, check and respond, line 1, using an IF statement to see if the device is not found. Remember to use the new variable name, in this example, device_one. If it finds the named device then print out a message, line 2. If it does not find the device, line 3, then print out a message to notify the user, line 4. Add these lines of code underneath the previous line. Save and run the program to find the two particular devices within your location.

```
    if (device_one != None):
      print “Linux Laptop is in”
    else:
      print “Linux Laptop is out”
```

17 Add an alternative action

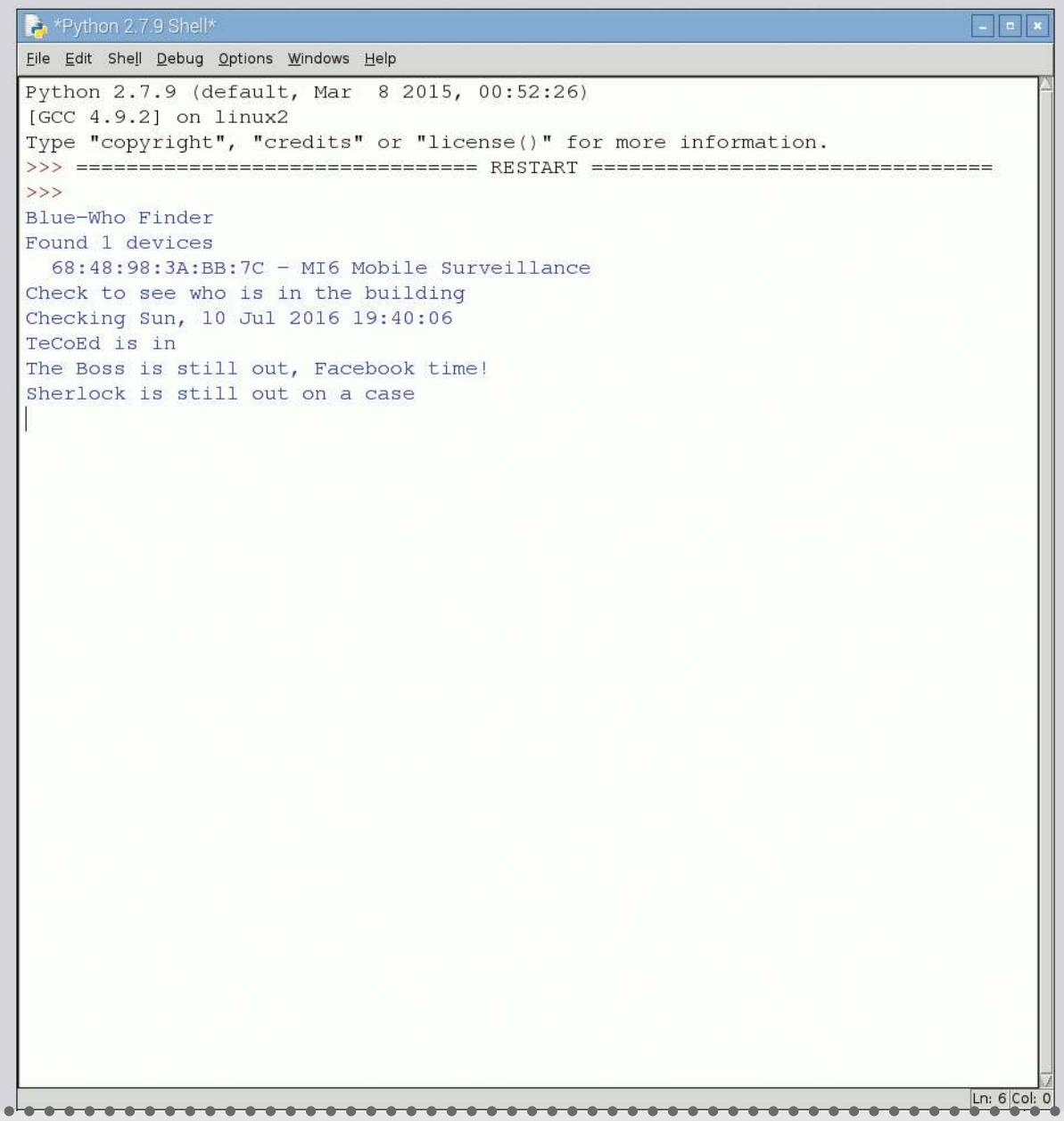
To customise the project further you could add your own action to the devices on being discovered. For example,

Variables

A variable is a location in the computer’s memory where you can store data. In order to find the data again you give the variable a suitable name or label. For example, days = 5. This means that the ‘days’ variable current holds the number five.

use a strip of LEDs to flash each time a device enters the location and is detected. Or use individual LEDs for each individual device to indicate if the device is present or not. How about combining the program with an LCD screen as a message board for who is present and who is out? For more inspiration, check out **https://www.youtube.com/watch?v=qUZQv87GVdQ**

```
  if (device != None):
    print “TeCoEd is in”
  else:
    print “Tecoed is out”
```



Hack a toy with the Raspberry Pi: Part two

Embed your hacks into your toy and create the code to bring it to life

In part one of this tutorial (see RasPi issue 34) you created four hacks that were originally used to augment a £3 R2D2 sweet dispenser making it light up, vibrate, play music and stream a live web feed to a mobile device. You may have been working on your own features to use and customise your toy. Part two of this tutorial begins by showing two different ways to set up and use a button to trigger your hacks. One method is to add and code your own button, the second method is to utilise the toy's own built-in button. The next part walks you through how to wire up, code and test each of the individual features before combining them into a single program which will bring your toy to life.

01 Prepare a button

A button is a simple way of triggering the hacks that you created in part one. Take a 4 x 6mm tactile button or similar and solder / attach a wire to each of its sides. Take each wire and connect it to one end of a female-to-female jumper wire. You can solder these into place or remove the plastic coating and wrap them around each metal contact.

FIND

MORE

FREE

MAGAZINES

FREEMAGS.CC

02 Set up the button

Next set up and test the button to ensure that it is working correctly. Take one of the wires and slot it onto GPIO pin 17, this is physical pin number 11 on the board. The second wire connects to a ground pin, indicated by a minus sign or the letters GND. The pin directly above GPIO pin 17 is a ground pin, physical pin number nine. This will complete the circuit and make the button functional.

03 Test the button

Open your Python editor and start a new file. Use the test program below to check that the button is functioning correctly. To ensure the buttons are responsive, use a pull up resistor code GPIO.PUD_UP, line 4. This removes multiple touches and means that only one 'press' is registered each time the button is pressed. Save and run the program. If it is working correctly it will return the message "Button works".

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.cleanup()
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)

while True:
    if GPIO.input(17) == 0:
        print "Button works"
```

04 Use the button on the toy

Instead of adding your own button you can utilise an existing button on the toy to trigger the events. On the R2D2 example this is the button at the front which releases the sweets and plays the classic R2D2 beep sound. Using a screwdriver

THE PROJECT ESSENTIALS

- An old/new toy
- Resistors
- Small disc motor
- LED
- A radio
- Small webcam
- Female-to-female jumper jerky wire
- Tactile button

or suitable tool, open the casing of your toy and then locate the electrics of the button. Locate the negative wire and cut this in two. Now attach a jumper wire to each of the ends. You can test that the connection is still working by joining the two jumper wires together and pressing the button.

05 Wire up your button

If your toy has a button to trigger a sound or movement then it will use batteries. These will still be used to power the toy but the extra circuit you added in Step 4 creates a secondary

circuit. When you press the button you join the two contacts and in turn complete the circuit, this sends a small current around the circuit which can be detected by the GPIO pin on the Raspberry Pi. Take one of the wires and attach it to GPIO pin 1, the 3.3 volts which will provide the current. Attach the other wire to GPIO pin 15, physical pin number 10. Pin 15 checks for a change in current. When the button is in its normal state, i.e. it has not been pressed, no current flows from the 3.3v pin as the circuit is broken. When you press the button it joins the contacts, the circuit completes and the current flows. Pin 15 registers a change in state which is used

to trigger an event.

```
cd /var/www
sudo chown pi: .
sudo rm *
wget http://wordpress.org/latest.tar.gz
```

The final step is then to extract the tarball, move all the files to the web server root and then go back to remove the original download. Do that with:

```
tar xzf latest.tar.gz
mv wordpress/* .
rm -rf wordpress latest.tar.gz
```

06 Test the button

Open a new Python file and enter the test code below. On line four a Pull Down is used to check for the change in state. The toy's button completes the circuit and GPIO pin 15 receives a little current. Its state becomes True or 1, line six, and it triggers the display of a confirmation message. The final line prints out a confirmation message each time the button is pressed.

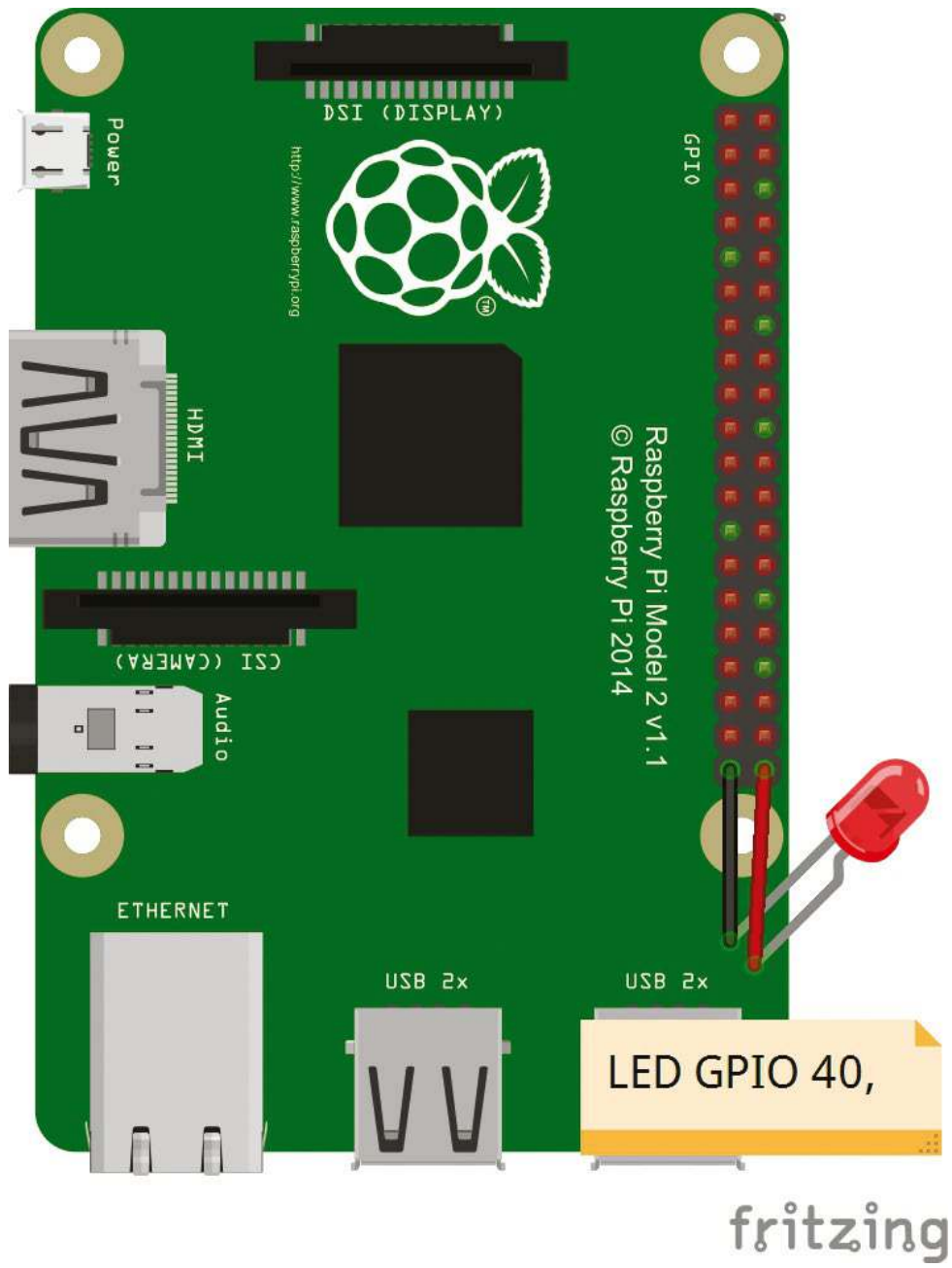
```
  import time
  import RPi.GPIO as GPIO
  GPIO.setmode(GPIO.BCM)
  GPIO.setup(15, GPIO.IN, GPIO.PUD_DOWN)
    #checks for a change on pin 7
  while True:
    if GPIO.input(15) == 1:
      print ("You touched R2D2!")
```

07 Wire up the LED

BCM Number

GPIO pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off. You can also program your Raspberry Pi to turn them on or off (output). The GPIO.BCM option means that you are referring to the pins by the 'Broadcom SOC channel' number. If you start counting the pins, this is the physical pin number. The GPIO.BOARD option specifies that you are referring to the pins by the plug number, i.e the numbers printed on the board

Now to connect the individual hacks to your toy. These examples are based on Part One of the tutorial but can be replaced with your own. Shut your Pi down and unplug the power. Assuming that the LED is connected to the jumper jerky wires, take the positive wire of the LED, the one with



the resistor, and attach it to GPIO pin 21. This is physical pin number 40, the one at the very bottom-right of the pins. The black wire connects to a ground pin. The nearest one is physical pin 39 just to the left of pin 40; attach it here.

08 Wire up the motor

Next take the positive wire from the motor, usually coloured red, and attach it to GPIO 09, which is physical pin number

21. You will then need to connect the other wire to any of the ground pins, (GND) 6, 9, 14, 20, 39. You may need to relocate this wire later on as you add more wires for the other components.

09 Add the PiFM aerial

The Raspberry Pi can broadcast to your radio directly from physical pin 7 without a need to alter anything. However, you will probably want to extend the range of the broadcast by adding a wire to GPIO 04, physical pin number 7. Unbelievably this can extend the range of the broadcast up to 100 meters. You must use physical pin number 7 to broadcast so move around any other wires that may be attached from your own hacks.

10 Add the web camera

The web camera is the simplest component to connect as it uses one of the USB ports. Once you have plugged it in use the command sudo lsusb to list the connections to the port. If it displays the name of your web camera, then it has been recognised and is ready to go. Consider stripping away the plastic shell so you are left with just the board and lens. Adjust the casing so that it fits neatly and can be hidden within your toy.

11 Setting up the program

Assuming that you have installed all the required software modules and libraries for your hacks, you are now ready to create the program to control your toy's extra features. Open a file in your Python editor and import os, this will control the PiFM and web camera, line 1. Next import the PiFM library; the webcam runs automatically when you boot up your Pi, see Part One in issue 168, Step 12. Set the GPIO pins to BCM on line 6 and then define the PUD for the button setup you

Check out R2D2 in action

Check out the video of the completed R2D2 toy hack and see the features in action. This may give you some ideas for your own toy hack. **https://www.youtube.com/watch?v=VnOsUaS5jSY**

are using. The first option, line 7, is for a button that is already connected to your toy, the second is to be used if you have added your own button.

```
import os
import sys
import time
import RPi.GPIO as GPIO
import PiFm
GPIO.setmode(GPIO.BCM)

GPIO.setup(15, GPIO.IN, GPIO.PUD_DOWN) #checks
  for a change on pin 7
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)
```

12 Set up the other outputs

Next prepare the two other GPIO outputs, in this example the LED and the motor. Set these as outputs using the code GPIO.setup(9, GPIO.OUT), line 1 and then turn the LED off using the code GPIO.output(21, GPIO.LOW). This ensures that when you start your program running the LED and the motor do not run until the trigger button is pressed.

```
GPIO.setup(9, GPIO.OUT)
GPIO.output(9, GPIO.HIGH)

GPIO.setup(21, GPIO.OUT)
GPIO.output(21, GPIO.LOW)
```

13 Set up the LED

Create a function to store the code that will control the LED, making it turn on for five seconds and then turn off again, lines 2, 3 and 4. Then set up a simple message to display at the start of the program to let you know that the toy is

gpiozero

This is a smart Python library that makes interaction with the GPIO pins really simple, for example you can control an LED with the code led.on(). It covers a massive range of components, hardware and add on boards. You can find out more here **https://gpiozero.readthedocs.io/en/v1.2.0/#**

ready and the pins are prepared. This is useful for debugging your program.

```
  def LED_Eye():
     GPIO.output(21, GPIO.HIGH)
     time.sleep(5)
     GPIO.output(21, GPIO.LOW)

  print (“Welcome to R2D2”)
```

14 Trigger the events

Set up a while loop to continually check if the button has been pressed, line 1. Use an IF statement to check when the button has been pressed and that the input is HIGH. This uses the line GPIO.input(15) == 1: In this case the 1 refers to an equivalent value of True or On; this relates to the circuit being completed and a current flowing through as discussed in Step 6. Then trigger the motor to turn on using GPIO.output(9, GPIO.LOW) line 5, and call the function LED_Eye() to execute, lighting up the LED, line 6.

```
  while True:
     if GPIO.input(15) == 1:
        print (“You touched R2D2!”)

        ‘’’Enable LED and Motor’’’
        GPIO.output(9, GPIO.LOW)
        LED_Eye()
```

15 Trigger the webcam and radio broadcast

Finally, start the web camera streaming using the line os.system(‘service motion start’) and check out the feed on your viewing device. While the webcam is running, start the

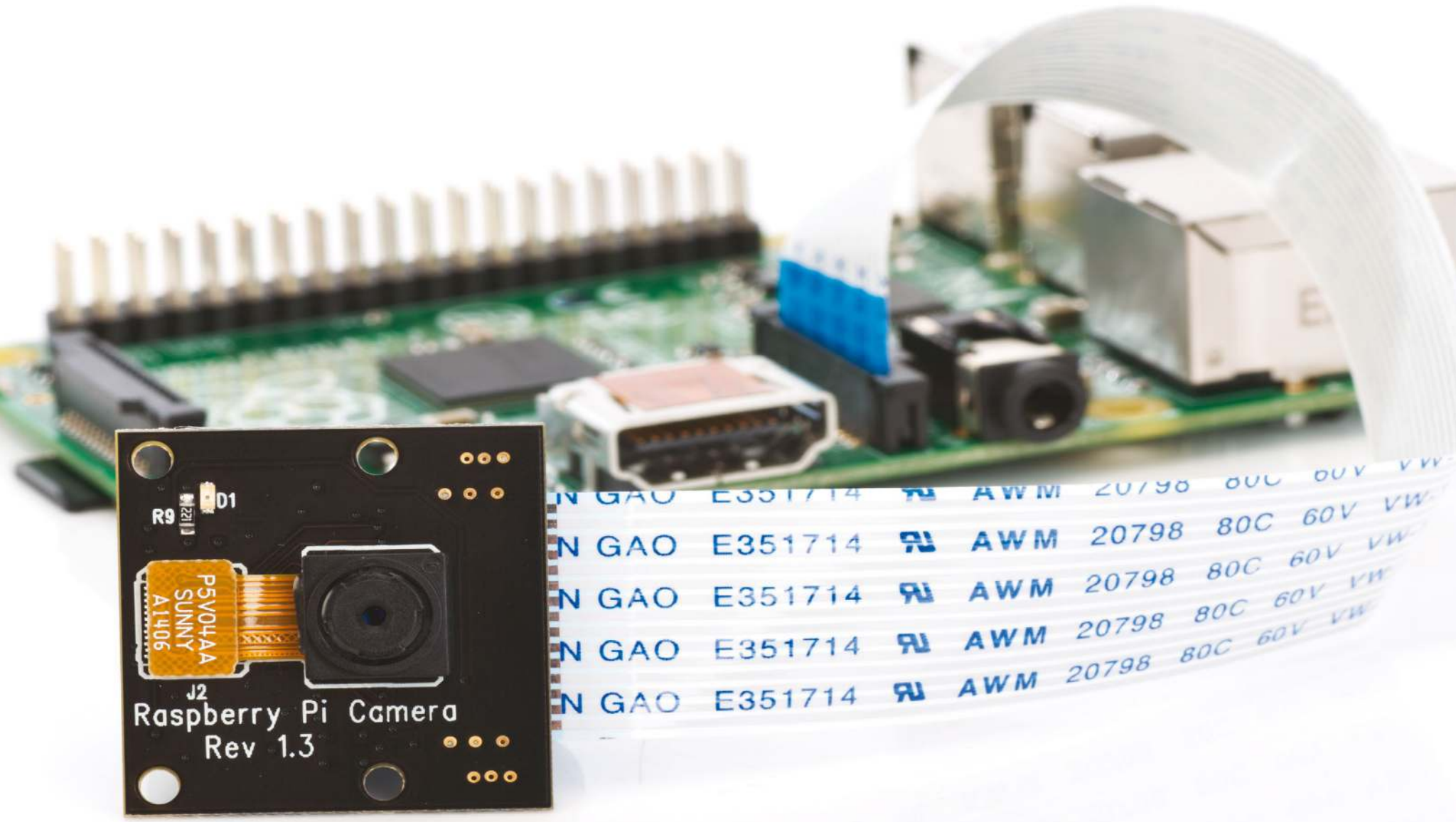
radio broadcast with PiFm.play_sound("sound.wav"), line 4. The default FM station is set to 100FM, tune in your radio to hear your sound being played. Then stop the web feed after the sound has finished using the line os.system('service motion stop'). Note that this code is indented on the same level as the previous lines.

16 Embed the hack into the toy

Once you have all your hacks triggering within the code, embed the wires and your Pi within your toy. You may choose to display the hardware and create a more 'augmented' style toy or discreetly hide it all away, surprising potential users when they interact with it.

Take night shots with the NoIR camera

Install the NoIR camera to enjoy some improved low light photographs and videos from your Raspberry Pi



Want to get more out of your Raspberry Pi's photographic abilities? You might have already connected a webcam or the official camera module, but found that certain activities – particularly shooting in low light – have been unsuccessful. The solution is a new camera module, the Raspberry Pi NoIR Camera Board, which has no infrared filter. This makes the camera ideal for taking infrared photographs or photographing objects in low light (you still won't be able to take photos in the dark without a light source). Capable of delivering 5MP still images, or recording video at 1080p and 30fps, the NoIR camera is easy to install - but for the best results, use with a case and portable power supply.

01 Check your contents

When you receive the Raspberry Pi NoIR Camera module, you'll find two items in the box. The camera module itself should be easy to spot, with its four mounting holes, on in each corner of the PCB. Look out also for the blue gel, a small square colour filter.

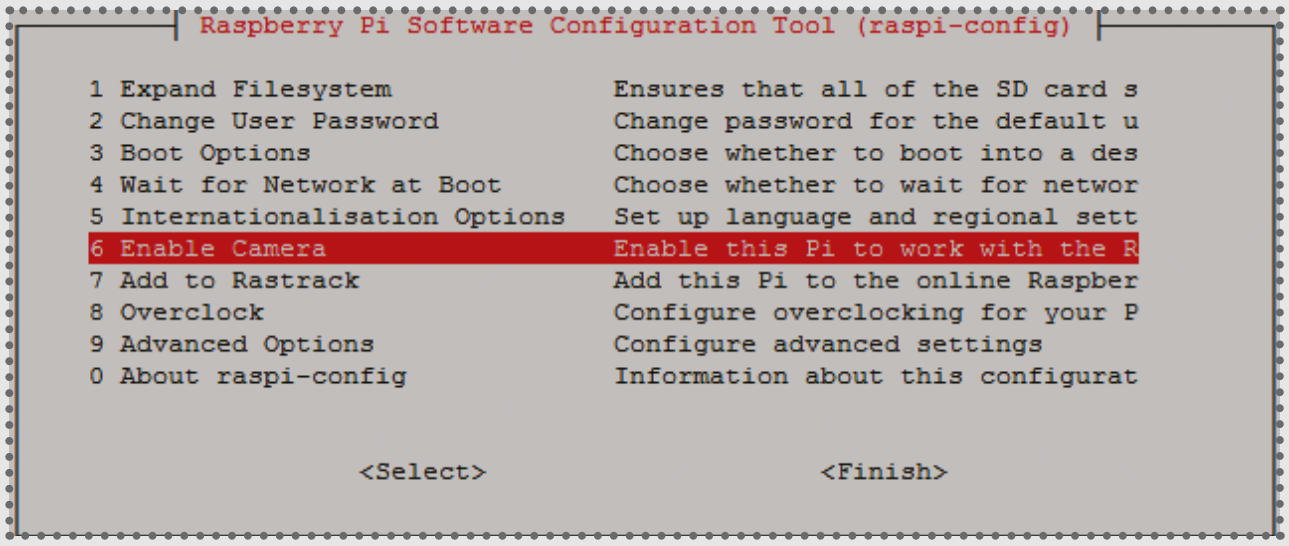
02 Connect the NoIR Camera module

Taking anti-static precautions, connect the NoIR Camera module to the Raspberry Pi using the ribbon cable, which slots into the ribbon connector with the silver side facing the nearby HDMI port.

Don't forget to lift the catch before inserting, and to press down firmly (without excessive pressure) to secure it.

03 Mount your camera module

If your Raspberry Pi case has a mounting point for the camera module, this will almost certainly have four small



Left After selecting Enable Camera, exit raspi-config - you'll then be prompted to restart

poles onto which the NoIR camera can be mounted. Simply line up the camera, lens facing the adequately-sized hole, and press the module into place to attach it.

04 Update your Pi

With the camera module safely attached, it's time to boot your Raspberry Pi and run the latest updates.

```
sudo apt install update
sudo apt dist-upgrade
```

should also open sudo raspi-config and Enable Camera, before restarting your Pi.

05 Take photos and videos

Two tools, raspistill and raspivid, are used for capturing stills and videos with the NoIR camera. Basic usage is as follows:

```
raspistill -o photo1.jpg
raspivid -o video1.h264
```

Use the help command with each to find the full set of commands, which include specifying image size and rotation (see the tip on the right).

```
pi@raspberrypi:~ $ raspivid --help
Display camera output to display, and optionally saves an H264 capture at reque
ted bitrate


usage: raspivid [options]

Image parameter commands

-?, --help      : This help information
-w, --width     : Set image width <size>. Default 1920
-h, --height    : Set image height <size>. Default 1080
-b, --bitrate   : Set bitrate. Use bits per second (e.g. 10MBits/s would be -b
0000000)
-o, --output    : Output filename <filename> (to write to stdout, use '-o -')
-v, --verbose   : Output verbose information during run
-t, --timeout   : Time (in ms) to capture for. If not specified, set to 5s. Zer
 to disable
-d, --demo      : Run a demo mode (cycle through range of camera options, no ca
ture)
-fps, --framerate       : Specify the frames per second to record
-e, --penc      : Display preview image *after* encoding (shows compression art
facts)
-g, --intra     : Specify the intra refresh period (key frame rate/GoP size). Z
```

Left Using your NoIR camera outdoors? Remember: your battery will drain more quickly when recording video

06 Take some test photos

To get the most out of the NoIR camera, you'll need to set it up to take some photos in low light. If you can't wait until the evening, you might use other low-light scenarios, such as an attic, basement, or cupboard.

For a test shot, try:

```
raspistill -o test.jpg
```

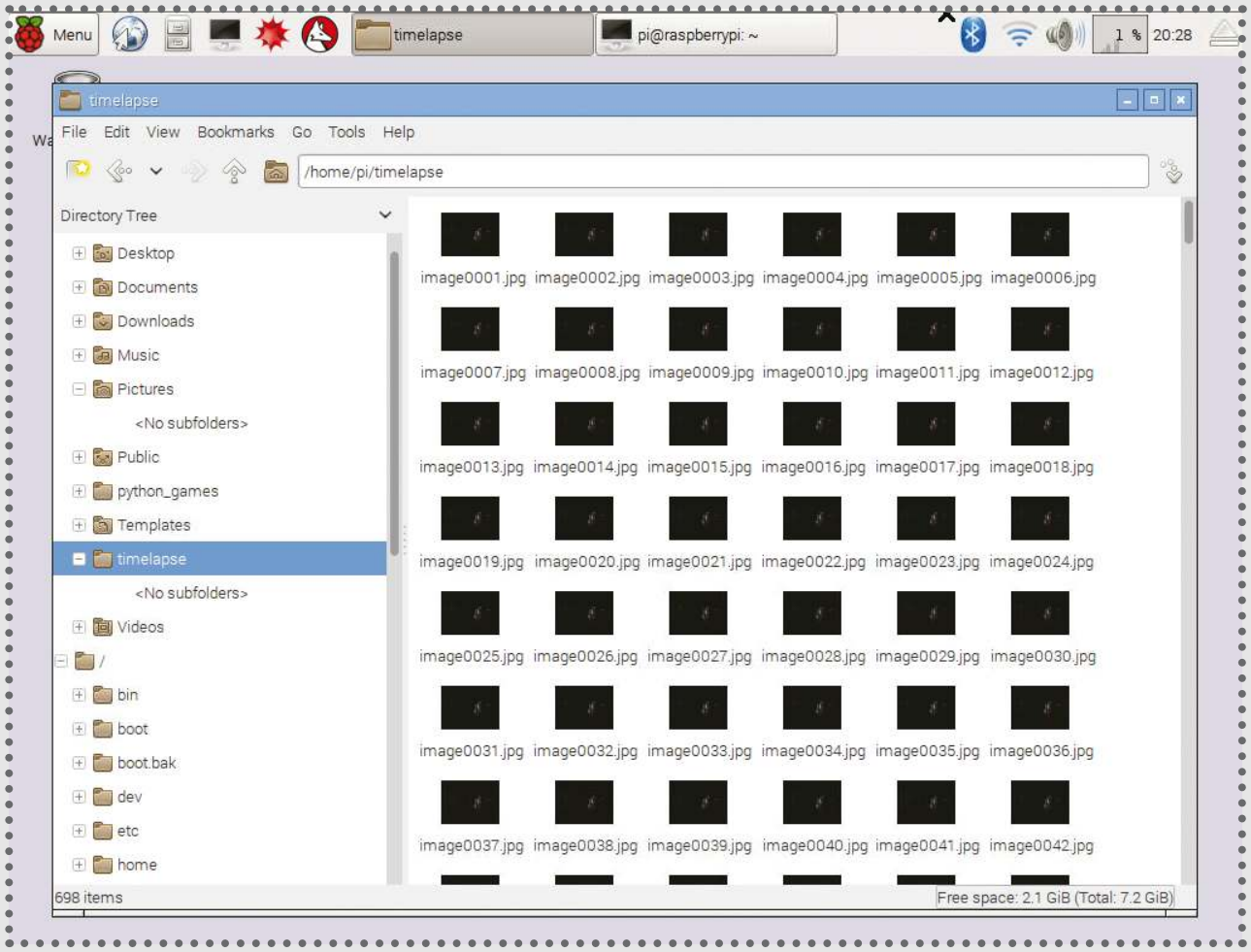
07 Prepare for an evening shoot

If you want to take it outdoors, you'll need a rechargeable battery pack to power the Pi and the camera. You should also have selected a relatively robust case, with good coverage to protect from the elements. For the best results, use an all-weather case, or employ a plastic bag!

08 Try time-lapse

Time-lapse photography is a good option for capturing images remotely (over SSH) without draining the battery.

```
raspistill -t 480000 -tl 5000 -o /timelapse/
  image%04d.jpg
```



Above Make sure you don't overload your SD card storage - keep your time-lapses brief!

Using millisecond values, this command snaps photos every 5 seconds for 8 minutes. Install libav-tools to convert your images into a movie.

06 Take some test photos

To compile the time-lapse images into a single video file, use:

```
avconv -y -f image2 -i /timelapse/image%04d.
jpg -r 24 -vcodec libx264 -profile high
-preset slow /timelapse/timelapse1.mp4
```

Once this compiles, use omxplayer to view the results:

```
omxplayer /home/pi/timelapse/
timelapse1.mp4
```

Set up a family calendar

Never forget an all-important birthday again by using your Raspberry Pi to display a calendar of events

It can be difficult to keep track of your grandparent's birthday or arranging to meet relatives, so this issue, we will look at how you can use a Raspberry Pi to display a family calendar and help you keep track of all those important dates using Python. As for the actual physical display, there are many options you could choose. These include small HDMI displays or small screens that plug directly into the GPIO pins. Your choice of display should be made based on where you want to mount the finished project. We will also be assuming that the calendars that we will be displaying from are Google calendars, as Google has a full Python library available that provides access to the API used to talk to the Google calendar system. If you want to use iCal or some other format, many of the concepts will be reusable but you will need to research the libraries for the specific calendar formats. Since we are going to be using online calendars, you want to add network connectivity, such as a Wi-Fi dongle, so that you can pull the data for the display. The first step will be installing the Python client with the command:

```
sudo pip install google-api-python-client
```

Google has been doing a lot of work tightening up the security of its APIs. Because of this, you need to go to Google and enable the API before you can use it. There is a wizard at **https://console.developers.google.com/start/api?id=calendar** that will walk you through the creation of a new project and the enabling of the API. You will then get a link to get the credentials for this project.

The page you get when you first click the Credentials button is a wizard where you can create a new credential. This is not what you need, however. In this case, you should click the Cancel button at the bottom of the page and then click on the OAuth Consent screen tab at the top of the screen to get set up. You can now set an email address and a product name and hit Save. You can now create a new OAuth client ID credential, setting the application type to Other. You also need to set a client name at this stage. Closing the resulting dialog box, you can download a JSON version of the client secret that will be used later on to authenticate with. You will likely want to rename the file to something easy to work with, like 'client_secret.json'. This JSON file contains details for authentication, including your client ID and a secret key. You need to keep this information secure.

Now the initial setup is taken care of, we can look at how to talk to the calendar server. Within the main googleapiclient module, there is a subsection called Discovery that will let you start the communication with the API of interest. Within this section, the starting point is a method called build. This code gives a boilerplate of opening communications with the Google calendar API.

```
import httplib2
from apiclient.discovery import build
from oauth2client import tools
from oauth2client.file import Storage
from oauth2client.client import
  AccessTokenRefreshError
from oauth2client.client import
  OAuth2WebServerFlow

client_id = 'client ID string'
client_secret = 'client secret string'
scope = 'https://www.googleapis.com/auth/
  calendar'
flow = OAuth2WebServerFlow(client_id, client_
  secret, scope)
storage = Storage('credentials.dat')
credentials = storage.get()
if credentials is None or credentials.
  invalid:
credentials = tools.run_flow(flow, storage,
  tools.argparser.parse_args())
http = httplib2.Http()
http = credentials.authorize(http)
service = build('calendar', 'v3', http=http)
```

This should give you a service object that you can use to query the calendar server to collect the details you need for your display. There is a lot of work contained in this code. The section where you create a storage object and get the credentials out of it is of particular note. The first time you do the check to see if you have valid credentials, it will fail and want to get a valid set of credentials. This

Why Python?

It's the official language of the Raspberry Pi. Read the docs at **python.org/doc**

is done by firing up a browser in order for you to go to Google Calendar to allow access. This means you will need to either have a GUI running on your Raspberry Pi this first time, or be using X11 forwarding if you are using SSH to connect to the Raspberry Pi.

So, now that we have an authenticated connection to the calendar server, we can start to work with the calendars and pull out the data that we need for the display. We are assuming that you will have multiple calendars to help organise your family; perhaps one for each person. If this is the case, the first step is to pull out a list of the available calendars for this account. You can do that by using the 'calendarList()' method. A full list is available with:

```
cal_list = service.calendarList().list().
  execute()
```

The returned data is a structured data set. You can get the entries for the individual calendars by using the keyword 'items'. The individual calendars have lots of information contained within them, such as a unique ID, a description and a summary, along with much more information. You could get a print-out of all of the summaries with this code.

```
for cal_item in cal_list['items']:
print(cal_item['summary'])
```

With a list of the available calendar IDs, you can start building up your list of the actual events that are going to populate your family calendar display. Each of these

events has a full dictionary of details, similar to the information available for the calendars. As a quick example, you can grab the summaries for all of the events for the primary calendar with the code here.

```
event_list = service.events().
  list(calendarId='primary').execute()
for event_item in event_list['items']:
print(event_item['summary'])
```

Using these examples, we could print off the summary, start time and date with:

```
for event_item in event_list['items']:
print(event_item['summary'] + ' ' + event_
  item['start']['dateTime'])
```

The format that the dateTime property gets printed in is along the lines of '2007-03-30T14:30:00-03:00'. So, you get the date, then a 'T', then the time in 24 hour format and then the timezone for the event. There is also an equivalent entry for the end of an event. This all works great, as long as a 'dateTime' property has been set for the given event. But you may have all-day events, so you will need to do some error checking when you start pulling this data out. If you entered a location for the event, that is stored in the 'location' property of the event that you can access in the same way that you pull out the summary. This is just the free-form text you entered when you created the event.

Some of the more important event details that you might be interested in include the list of attendees and a list of any included attachments. Both of these are lists of structured data. For each event, you could loop through

"You need to go to Google and enable the API beforehand"

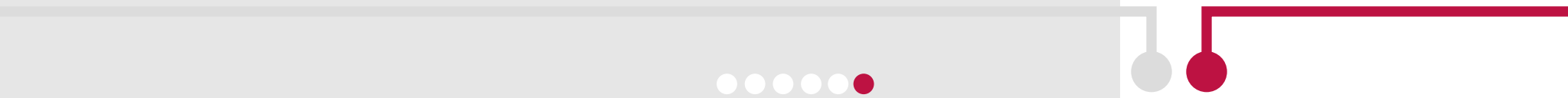
and pull out the name of each attendee with this code.

```
  for attendee in event_item['attendees']:
  print(attendee['displayName'])
```

Items like the 'displayName' are optional, so you want to be sure that you actually get a string back before trying to do anything with it. In a similar way, you could loop through attached files to a given event.

```
  for attachment in event_items['attachments']:
  # Do something interesting
```

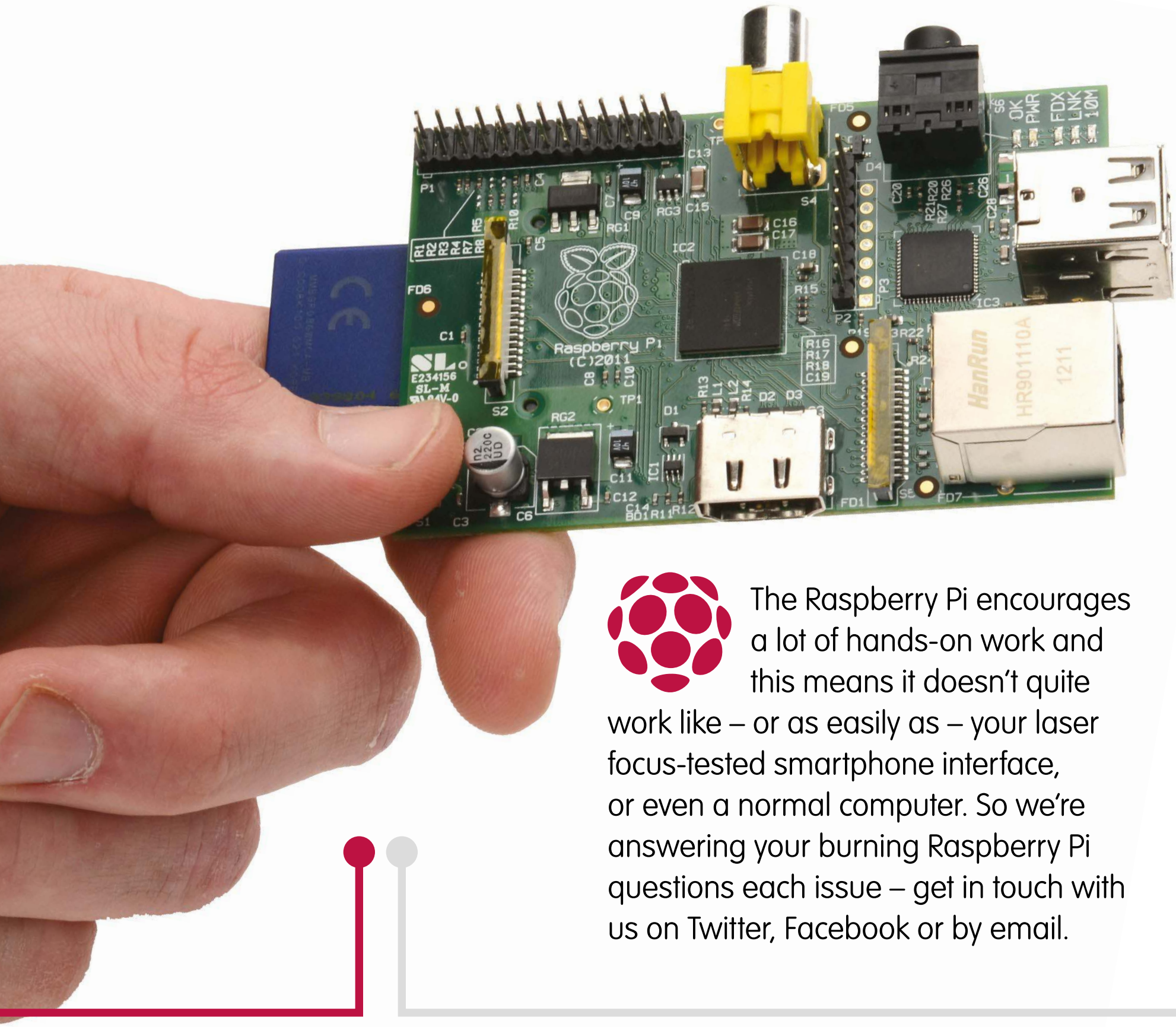
This is a bit messier if you wanted to pull any extra detail from the attachments. To begin with, you will likely want to look at the 'title' and the 'mimeType' properties to see if there is anything useful that you can do with it within your calendar display. If there is, the attachment will show up in one of two ways. If it is stored in Google Drive, there will be a 'fileID' property that you can use within the Google Drive API to pull the file information down. If it is a file located somewhere else, there will be a 'fileUrl' property that you can use to pull the file down from.



Talking Pi

Join the conversation at...

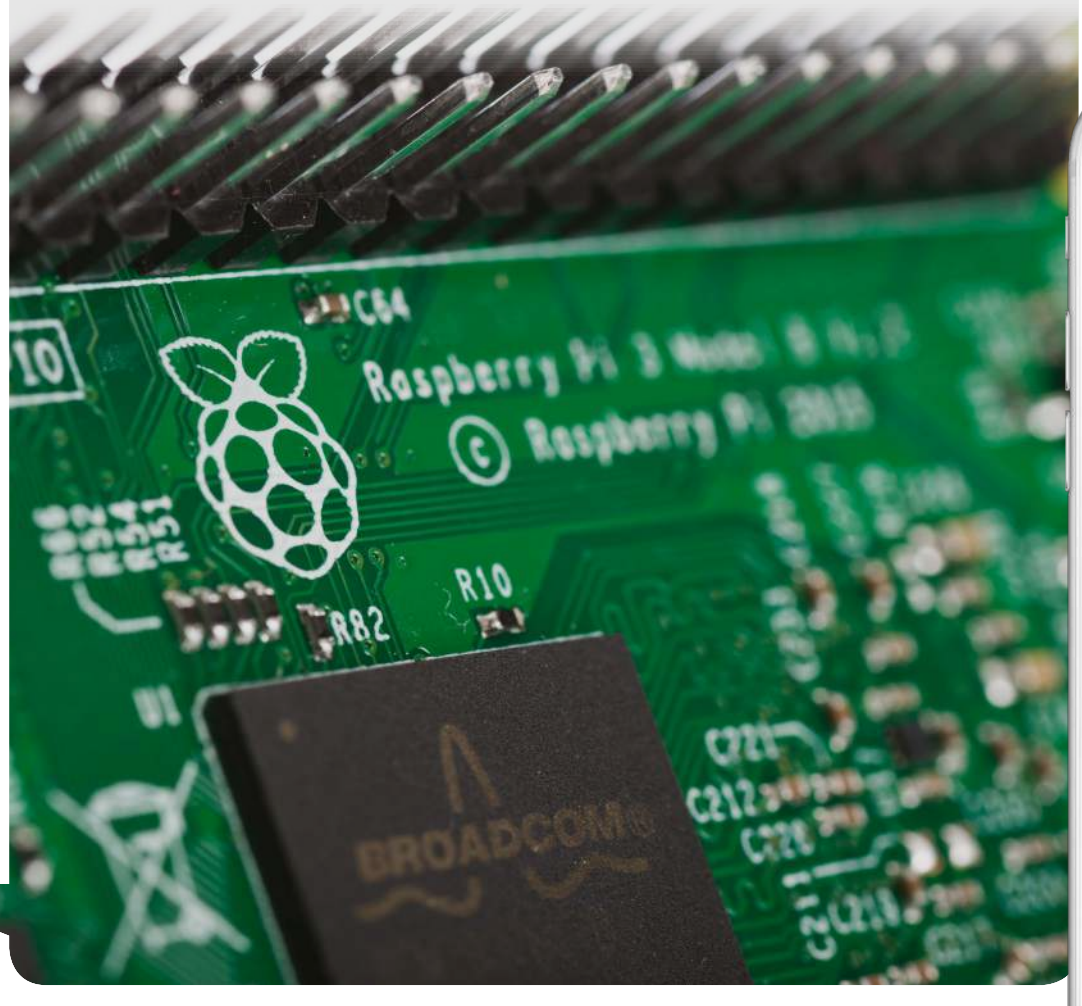
@linuxusermag | Linux User & Developer | linuxuser@futurenet.com



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easily as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

I've just got into the Pi but something confuses me. Why do I keep hearing it called an SoC or SBC?
Cillian via email

The world of tech is full of acronyms and it can be difficult keeping up with them all, especially if you're new to the Pi and its ilk. The two terms are very similar in meaning, and both describe what the Pi is and how it works. 'SoC' means 'System on a Chip' and refers to the fact that the Pi's processor does a lot more of the grunt work than the CPU in a computer (relatively speaking; unless your PC is really old then it's got a lot more power than the Pi). The chip is responsible for a lot more of what the Pi does. 'SBC' means 'Single Board Computer', and refers to the fact that all the Pi's essential hardware lives on one tiny board!



Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

Should I get a Raspberry Pi or an Arduino for my projects?
Jo via email

Call us biased, but we're always going to say "Get a Raspberry Pi!" Well, we would, wouldn't we? The thing is though, it's not so much a one-or-the-other question and more a question of what you want your project to do (or what you want to do with it). Both boards have their advantages, but the best thing is that the Raspberry Pi and the Arduino can work together. Take a look at the Pi Project this issue for a great example of a maker project that combines the two!

Why can't I add more RAM onto my Raspberry Pi? I have a 2B.
David via email

The whole point of an SBC is that it should have everything that you need on board already, although there are people out there like yourself who'd like more memory available for their projects. Unlike earlier models, the 2B and 3B's RAM is on separate chips on the bottom of the board, so looks temptingly like it can be upgraded. Unfortunately though, the Model 2B can't handle any more than 1GB of RAM. This is the most that the Pi's processor can handle; any more and at best you're likely to see all sorts of errors popping up, while at worst you'll seriously damage your Raspberry Pi!

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

Control your Pi using Amazon Echo

Get this issue's source code at:
www.linuxuser.co.uk/raspicode

9021

9000